

From Logs to Logic:
Learning and Evaluating Interpretable Representations
of Behavior for Autonomous Systems

Örebro Studies in Technology 108



Simona Gugliermo

From Logs to Logic:
Learning and Evaluating Interpretable
Representations of Behavior for
Autonomous Systems

© Simona Gugliermo, 2025

Title: From Logs to Logic:
Learning and Evaluating Interpretable Representations of Behavior for
Autonomous Systems

Publisher: Örebro University, 2025
www.oru.se/publications

Printer: Örebro University/Repro 10/2025

ISSN 1650-8580
ISBN 978-91-7529-706-4

Abstract

Autonomous systems are increasingly being deployed across various real-world domains, such as fleets of self-driving vehicles, robotic warehouses, and delivery services using unmanned aerial vehicles. These systems are required to operate with high reliability and predictability, to adapt continuously to changing conditions, and to remain accountable to human supervisors. To achieve these objectives, autonomous systems need explicit, formal representations of their behavior that facilitate task planning, system verification, and human oversight.

In current industrial practice, such representations, whether for task-level control or action planning, are typically engineered manually. While hand-crafted representations can be precise, their development is labor-intensive and difficult to scale. Learning-based approaches offer a promising alternative by extracting behavioral representations from execution data. However, they often make unrealistic assumptions, such as access to simulated environments or large volumes of high-quality training data. Moreover, they fail to simultaneously achieve all the three critical objectives, that is reliability, adaptability, and interpretability. Therefore, there is a clear need for methods capable of efficiently learning accurate, interpretable representations under realistic conditions.

In this thesis, we address the problem of learning interpretable representations of system behavior from execution traces - sequences of observed actions and state transitions generated during the operation of autonomous systems. Learning from such traces is appealing because they are readily available from system logs and provide direct evidence of how a system behaves in realistic, often complex environments. The overarching goal is to derive representations that not only support automated planning but also enhance human understanding and oversight.

Two distinct types of behavior representation are explored: Behavior Trees (BTs) and STRIPS-style planning domains. For each, a novel method to automatically construct representations from execution traces is proposed. Specifically, for BTs, we introduce a method that combines Boolean logic, leveraging algorithms originally developed in circuit theory, with decision tree learning to induce structured, interpretable behavior representations. To assess the in-

terpretability of BTs, a user study is conducted to examine how such representations are perceived by human users. The study identifies key features that influence user comprehension, contributing empirical evidence to a space that has traditionally lacked systematic analysis. Furthermore, a structured evaluation method for BTs along with quality metrics and design principles is presented, addressing the current lack of guidance for assessing BT quality beyond functional performance.

For STRIPS-style domains, we introduce a novel learning framework to construct symbolic action representations directly from execution traces, even in the presence of noise. In addition to the learning algorithm, a systematic methodology is proposed for evaluating learned planning domains through structural and task-based analysis, thereby addressing a critical gap in current practice and thus responding to the growing need for rigorous assessment methods.

The results demonstrate that it is possible to extract interpretable representations of autonomous behavior from noisy data. The proposed methods enable the transition from raw execution traces to structured representations that can support planning, validation, and human-in-the-loop systems. By advancing methods for learning, interpreting, and evaluating learned behavior representations, this work contributes to the development of autonomous systems that are both operationally effective and intelligible to human stakeholders.

Acknowledgements

This has been a very long and intense journey where many actors have played a part. Each of them deserves my gratitude, because everyone taught me something, and in the end, this is what truly matters.

Thank you to Federico and Chris, who first conceived this project and made it possible. You wrote the proposal and set me on this path by sharing your vision and valuable ideas.

Thank you to Erik and Alessandro, who stepped in when I felt completely lost. Your support since then has been constant, and I have always felt that you genuinely cared about me as a person, while truly wanting me to succeed in this.

Thank you to Uwe, the last to join, but whose technical expertise was deeply valuable and sincerely appreciated.

Thank you to the entire AASS team, particularly Todor and Jenny, for your support and help. I am especially grateful to the CRS Lab, who supported me throughout this journey and treated me as one of their own despite my hybrid position.

Thank you to Scania for giving me the opportunity to begin this journey. It is not common in many countries to find companies willing to support their employees in pursuing academic studies. I am grateful to the managers who stood by me along the way: Jonatan, Sandra, and last but certainly not least, Mariette.

Thank you to the old Fleet Control Team, where everything started, and to the AI Enablement Team, where I am now, as everything comes to an end. I could not imagine a better place to be, nor better people to share this last stretch with.

Alongside the professional path, I also met incredible people who became part of my personal life. In particular, thank you to Marco, David, and Edu. Thank you for the laughs, the struggles, and the help (and the gossip). You made every trip to Örebro lighter and more fun. It simply would not have been the same without you.

A special acknowledgment also goes to Who quietly helped me shape words and ideas when clarity was most needed. Without this silent support, everything would have been much harder.

Grazie a Mamma, Papà e Stefania. Non mi avete mai fatto sentire sola o lontana, seppur un po' lontana. Mi avete sempre spinto ad andare oltre, insegnandomi che i limiti più grandi sono quelli che ci imponiamo da soli. Impagabile il vostro supporto e la fiducia che riponete in me ogni giorno.

Grazie a Stefano, che mi ha letteralmente raccolta e mi ha accompagnata verso la fine di questo percorso con tanta pazienza e tante, tantissime risate. Grazie per aver condiviso con me i giorni pieni di dubbi, i weekend di lavoro, le lacrime improvvise e persino le mie risposte acide. Grazie per avermi ascoltata e rassicurata, per avere sempre creduto in me e per avermi spinto a fare lo stesso. Grazie perché insieme abbiamo imparato a vedere la luce alla fine del tunnel. Sempre.

Infine, vorrei ringraziare la me di prima, che è diventata la me di oggi. Forse sono stata l'unica vera costante in questo cammino fatto di variabili e incertezze. Per tutte le volte in cui ho pensato di non farcela... e invece ce l'ho fatta.

Contents

1	Introduction	1
1.1	Motivation	1
1.1.1	From Data to Action Models	2
1.1.2	Industrial Motivation	2
1.1.3	Problem Scope and Research Focus	3
1.2	Research Objective	4
1.3	Proposed Approach	5
1.3.1	BTs as Intermediate Representations	5
1.3.2	Direct Domain Learning with BT Support	6
1.3.3	Validation	7
1.4	Contributions	8
1.5	Outline	9
1.6	List of Publications	10
2	Background	13
2.1	Boolean Logic Representations	13
2.1.1	Fundamental Concepts	14
2.1.2	Algebraic Trees and Factorization	14
2.1.3	Boolean Algebra	14
2.2	Decision Trees	16
2.2.1	Learning of Decision Trees	18
2.3	Behavior Trees	20
2.3.1	Classical Behavior Tree	21
2.3.2	Non Classical Behavior Trees	23
2.4	AI Planning	24
2.4.1	Classical Planning	25
2.4.2	Non Classical Planning	30
3	Learning Behavior Trees	33
3.1	Related Work	33
3.1.1	LfD-Based BT Learning	34
3.1.2	Learning Compact BTs Using DTs	35

3.2	BT-Factor	36
3.2.1	Overview	37
3.2.2	Rule Extraction	39
3.2.3	Rule Conversion	40
3.2.4	Rule Factorization	41
3.2.5	BT Conversion	43
3.3	Experimental Set Up and Results	44
3.3.1	Setup I: Industrially-Relevant Simulation Environment	44
3.3.2	Setup II: Synthetic Data Simulator	47
3.3.3	Setup III: Pick-and-Place Demonstrations	48
3.4	Summary and Outlook	51
4	Evaluating Behavior Trees	53
4.1	Related Work	54
4.2	Metrics	55
4.2.1	Functional Metrics	55
4.2.2	Non-Functional Metrics	57
4.3	Properties	58
4.3.1	Functional Properties	58
4.3.2	Non Functional Properties	62
4.3.3	Design Recommendations	65
4.3.4	Interrelationship among Properties	66
4.4	User Study on Interpretability	66
4.4.1	Experimental Protocol	67
4.4.2	Hypothesis	71
4.4.3	Sample	71
4.4.4	Preliminary Analysis	72
4.4.5	Hypothesis Testing	74
4.4.6	Discussion of the Findings	78
4.5	Summary and Outlook	80
5	Learning Planning Domains	81
5.1	Related Work	82
5.1.1	Learning from Deterministic Plan Traces	82
5.1.2	Learning with Numerical State Variables	84
5.1.3	Learning from Noisy Observations	85
5.2	LAML	86
5.2.1	Data Preprocessing	88
5.2.2	Prelifting	90
5.2.3	Learn Lifted Preconditions	91
5.2.4	Learn Lifted Effects	94
5.2.5	Generate PDDL Domain	96
5.3	Comparison with NOLAM	97
5.4	Summary and Outlook	97

6	Evaluating Learned Planning Domains	99
6.1	Related Work	100
6.2	Evaluation Methods	101
6.3	Evaluation Metrics	103
6.4	LAML Evaluation	107
6.4.1	Comparison with Reference Domain	109
6.4.2	Plan Generation Feasibility	110
6.4.3	Comparison with Historical Plans	111
6.4.4	Plan Validation	112
6.5	Summary and Outlook	113
7	Discussion	115
7.1	Topics for Discussion	115
7.2	Behavior Trees vs Planning Domains	117
7.3	Converting Behavior Trees to Planning Domains	119
7.4	Humans and Planning Domains	122
7.5	Applicability to Real-World Scenarios	125
8	Conclusions	129
8.1	Summary of the Contributions	129
8.2	Industrial Relevance	132
8.3	Limitations and Assumptions	133
8.4	Ethical, Social, Economic impact	134
8.5	Future Work	136
	References	139

List of Figures

1.1	Initial two-phase approach with BTs as intermediates	6
1.2	Final approach with direct model learning and BT interface . . .	7
1.3	Block diagram of chapters, contributions, and research questions	9
2.1	Algebraic tree factorization	15
2.2	Example of a decision tree.	16
2.3	Decision tree associated with dataset in Table 2.1.	17
2.4	Example of a Behavior Tree	21
3.1	Overview of the BT-Factor approach for learning Behavior Trees	37
3.2	Battery management scenario used to illustrate BT-Factor . . .	39
3.3	Comparison of Behavior Trees after applying BT-Factor.	43
3.4	Simulated environment for Setup I.	44
3.5	Impact of GFACTOR on node count and condition-fallback ratio.	49
3.6	Effect of noise on BT-Factor accuracy and rule complexity. . .	50
4.1	Overview of BT design pattern examples.	70
5.1	Input-output mapping for PDDL action model learning	81
5.2	Evolution of the LAML learning framework	87
5.3	LAML: Illustrative example of data preprocessing	90
5.4	LAML: Illustrative example of prelifting	92
5.5	LAML: Illustrative example of preconditions learning	94
5.6	LAML: Example of learned PDDL domain	96
6.1	Precision and recall across noise levels	109
6.2	Problem resolution success rate across noise levels	110
6.3	Plan comparison success rate across noise levels	111
6.4	Validation success rate across noise levels	112

List of Tables

2.1	Example dataset for decision tree learning.	17
3.1	Overview of BT-Factor core modules	38
3.2	Symbols mapping in battery management example	43
3.3	BT-Factor: Correctness experiments	46
3.4	BT-Factor: Performance experiments	46
3.5	BT-Factor: GED experiments	47
4.1	Mapping of BT evaluation metrics and properties	64
4.2	Design recommendations for BT properties	65
4.3	Distribution of answers for familiarity with BTs	72
4.4	Aggregated BT familiarity variable	72
4.5	Test for the correlation confidence - response correctness	73
4.6	Confidence vs. response correctness, low familiarity	73
4.7	Confidence vs. response correctness, medium familiarity	74
4.8	Confidence levels of highly BT-familiar participants	74
4.9	BT design vs. response correctness	75
4.10	BT design vs. confidence	76
4.11	Categorization of the width-to-height ratio	76
4.12	Width-to-height ratio category vs. response correctness	77
4.13	Familiarity with BT vs. response correctness	77
4.14	Familiarity with BT vs. confidence	78
6.1	Comparison of evaluation methods for learned domains.	106
6.2	PDDL domain specifications	108
6.3	Precision and recall metrics for different noise levels	109
7.1	Comparison between BT and planning domain	120

List of Algorithms

1	C5.0 decision tree algorithm	19
2	Factorization of a Boolean expression	42
3	Data preprocessing state variables	89
4	Data preprocessing actions	89
5	Prelifting state variables	91
6	Learn lifted preconditions	93
7	Learn lifted effects	95

Chapter 1

Introduction

1.1 Motivation

Autonomous systems, such as self-driving vehicles, drones, or industrial robots, are becoming increasingly integrated into a wide range of applications, from transportation and logistics to manufacturing and environmental monitoring. Depending on the application, such systems may range from semi-autonomous, requiring human supervision, to fully autonomous, operating independently by leveraging sensors, data processing, and algorithmic reasoning to perceive and act upon their environment. A key enabler of autonomy in such systems is their ability to make decisions about which actions to perform in order to achieve given objectives. Within the field of artificial intelligence, this capability is often achieved through automated planning [52], which involves constructing plans based on formal representations of the environment, available actions, and intended goals. Planning capabilities allow autonomous systems to generate behavior that is not hard-coded in advance, but instead flexibly derived from symbolic representations of the possible actions. These representations specify the preconditions that must be satisfied for each action to be applicable, as well as the effects that describe how the environment changes once the action is executed. This structure enables machines to compute valid plans using general-purpose algorithms.

Despite their expressive power and theoretical elegance, action representations are typically crafted manually. This process involves identifying relevant state variables, defining action schemas, and encoding them in a formal planning domain language. As a result, it requires significant domain expertise and is both time-consuming and error-prone, particularly in dynamic environments, where models must be frequently revised to reflect changing operational realities [84]. The difficulty of constructing high-quality domain models is widely acknowledged in the planning literature [166].

1.1.1 From Data to Action Models

To address the aforementioned challenges, a growing body of research has explored the application of machine learning techniques to automatically extract action representations from data [83]. In many autonomous systems, rich operational data is routinely logged during execution in the form of execution traces, chronological records of system states and actions taken. These traces provide a valuable source of information for inferring the underlying structure of system behavior.

However, the challenge extends beyond merely learning action representations. According to the knowledge representation hypothesis [15], a representation must serve a dual function: it must act both as a medium for computational manipulation and as a vehicle for conveying semantic content that is interpretable by humans. Consequently, action representations should satisfy this dual requirement. In the context of planning, this means that representations must not only support the correct generation of plans but also facilitate human understanding, debugging, and refinement, for instance, through visualization techniques [10]. This is particularly important in safety-critical or human-in-the-loop settings, where system transparency and interpretability are essential. Ensuring that learned representations are intelligible to human users is thus a key prerequisite for building trustworthy and accountable autonomous agents capable of operating at scale.

These challenges highlight a fundamental gap in the current state of the art: while significant progress has been made in automating the generation of plans from symbolic models [7], the acquisition of these models themselves, particularly in ways that ensure interpretability, remains a major bottleneck. This thesis aims to address this gap by investigating how action representations can be learned from data in a manner that supports both automated reasoning and human understanding.

1.1.2 Industrial Motivation

This research is motivated by a concrete industrial need identified in collaboration with Scania, a global manufacturer of sustainable transport solutions. Scania’s vision for autonomous transport goes beyond self-driving vehicles, encompassing intelligent offboard decision-making for logistics coordination, vehicle-task allocation, and overall fleet orchestration¹. In such systems, scalable and adaptable planning capabilities are key to operational efficiency and system robustness, as they enable the system to dynamically respond to changing demands, disruptions, and resource constraints across the fleet.

This example serves as an illustration of the broader problem discussed above: autonomous fleet management requires planning models that are both

¹<https://www.scania.com/group/en/home/innovation/autonomous-solutions/hub-to-hub.html>

effective for automated reasoning and intelligible to human stakeholders. Systems must operate across diverse application domains such as mining, urban delivery, and long-haul transport, each with its own unique operational characteristics. Manually constructing or adapting symbolic representations for each context is both labour-intensive and difficult to scale. The ability to automatically learn how systems behave, that is, to infer the underlying structure of available actions and their interactions from observed operations, offers a promising alternative. Crucially, however, these learned representations must remain interpretable and verifiable, so that system designers can understand, validate, and, when necessary, refine them.

1.1.3 Problem Scope and Research Focus

This thesis addresses the problem of **learning abstract action models from instances of execution traces**, where system behavior is captured in logged sequences of state transitions and actions. The objective is to extract interpretable symbolic models from these traces, making it possible to reason about system behavior, generate new plans, and support autonomous decision-making without requiring hand-crafted models.

This work brings together insights from two different scientific communities: the planning community, which emphasizes symbolic models with declarative semantics, such as PDDL domains [113], and the robotics community, which often relies on procedural, interpretable structures such as Behavior Trees (BTs) [29] to encode agent behavior. Each approach brings distinct advantages and limitations. PDDL-based models are powerful representations for decision-making, but difficult to construct and validate. BTs, on the other hand, are widely used in robotics for their modularity and clarity. However, despite their advantages in terms of modularity and interpretability, BTs lack the declarative semantics and explicit precondition-effect structure that automated planning frameworks typically rely on for plan synthesis and verification.

This thesis explores how insights from both communities can be used to address the challenges of learning behavior models for fleet management applications. Specifically, it investigates how symbolic models can be induced from execution traces in a way that remains interpretable, domain-adaptive, and suitable for human-in-the-loop validation. To this end, the thesis focuses on three interrelated research challenges:

1. Bridging the abstraction gap between execution traces, which capture system behavior as low-level, temporally ordered sequences of states and actions, and symbolic planning representations, which describe behavior in terms of high-level, declarative action models suitable for reasoning and generalization.

2. Supporting human-in-the-loop workflows, where existing experts can inspect, guide, and debug the learning process and inject their existing knowledge.
3. Ensuring interpretability and transparency, so that human operators and engineers can understand, validate, and refine the learned models.

The methods developed are not tailored to a specific application, but they are designed to be generalizable across domains where execution traces are available and planning is beneficial. While the work is motivated by Scania's need for scalable autonomy in fleet management, it also contributes to a broader challenge in artificial intelligence: learning the action representations required for automated planning and acting directly from observed system behavior. As such, our work contributes to the development of adaptive and interpretable planning systems that can be deployed across diverse operational contexts where hand-crafted models are infeasible or insufficient.

1.2 Research Objective

Overall Objective develop and evaluate methods for learning structured, symbolic representations of system behavior from execution traces, with a particular emphasis on interpretability, verifiability, and human-centered integration.

This objective is pursued through the following four complementary research questions:

RQ1 How can interpretable representations of system behavior be learned from execution traces in autonomous environments?

This question focuses on the core learning problem: extracting meaningful, structured abstractions from sequential data collected during system operation, in a way that captures relevant patterns for planning and reasoning.

RQ2 What forms of representation best support interpretability and facilitate human understanding of the system behavior?

This question focuses on representational design: identifying symbolic structures that are not only functionally adequate for planning but also transparent and accessible to human stakeholders such as engineers and fleet management experts.

RQ3 How can human users be effectively integrated into the process of validating, correcting, and refining learned representations?

This question addresses the design of human-in-the-loop workflows, where expert knowledge and feedback are used to validate learning and ensure alignment with operational realities.

RQ4 What metrics and methods can be used to evaluate specific properties of the learned representations?

This question investigates how to assess the quality and practical utility of the symbolic representations by identifying relevant properties, such as correctness, completeness, robustness, and interpretability, and selecting or designing appropriate metrics and evaluation methods to ensure that the representations are not only formally sound but also useful in realistic deployment scenarios.

1.3 Proposed Approach

The approach adopted in this thesis is shaped by a dual objective: first, to learn a symbolic representation that can be used for automated planning, such as a PDDL domain model; and second, to ensure that the learned model remains interpretable and accessible to human stakeholders, in the spirit of behavior representations such as BTs. This dual aim calls for a learning architecture in which declarative planning models and procedural structures can be integrated in a coherent and complementary way. Over the course of the research, the role of BTs within this architecture underwent significant revision. Early experiments explored the use of BTs as an intermediate representation from which symbolic models could be derived. However, as the work progressed, this initial pipeline was adapted to instead learn symbolic models directly from execution traces, while retaining BTs as a mechanism for human-centred interpretation and interaction. This evolution in approach reflects a deeper understanding of the trade-offs between symbolic expressiveness and procedural interpretability, and led to a revised framework outlined in the remainder of this chapter.

1.3.1 BTs as Intermediate Representations

The approach initially adopted in this research was a two-phase pipeline:

1. **Intermediate Representation Learning:** Extract an interpretable structure (a BT) from execution traces that captures the expert decision patterns and control flow logic.
2. **Symbolic Model Derivation:** Transform the interpretable representation into a formal planning domain comprising action models defined in terms of symbolic preconditions and effects.

The justification for this approach was that BTs reduce the abstraction gap between raw data and symbolic representations, provide a tangible interface for human validation, and finally, they provide a pathway to symbolic models suitable for automated planning. Figure 1.1 illustrates this pipeline.

While this approach showed promise, it also revealed significant limitations. The transition from BTs to planning domains involves significant conceptual

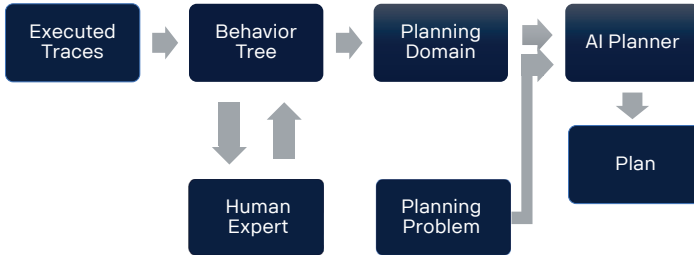


Figure 1.1: Overview of the initial approach: a two-phase pipeline where Behavior Trees (BTs) are first learned from execution traces as interpretable procedural structures, and then used as an intermediate representation to derive symbolic planning domains. While this approach supported human interpretability, it faced significant challenges in extracting generalisable, causally grounded action models from control-oriented BT structures.

and technical challenges. Standard BTs, by design, focus on execution control flow rather than declarative semantics. They represent when and how behaviors should be executed, but do not encode states or causal relationships. By contrast, action models in planning domains explicitly define such elements through symbolic preconditions and effects, enabling reasoning about causality and goal-directed behavior. Consequently, inferring a correct and complete planning domain model from a BT required additional assumptions and complex causal inference techniques.

These challenges revealed fundamental limitations in the initial pipeline and motivated a shift toward a revised approach that learns symbolic action models directly from execution traces, without relying on intermediate procedural representations. This revised approach rests on the assumption that general causal relationships, essential for constructing planning models, can be inferred from the original traces themselves, whereas such information is compiled away in the learned BTs.

1.3.2 Direct Domain Learning with BT Support

The final approach adopted in this thesis, shown in Figure 1.2, reverses the initial pipeline. Instead of using BTs as intermediates, the system now learns planning domains directly from trace data, while retaining BTs as interfaces for human-centered interpretation and validation.

In this revised approach, BTs are no longer used to construct symbolic models but to visualize and communicate learned behaviors. They serve as a

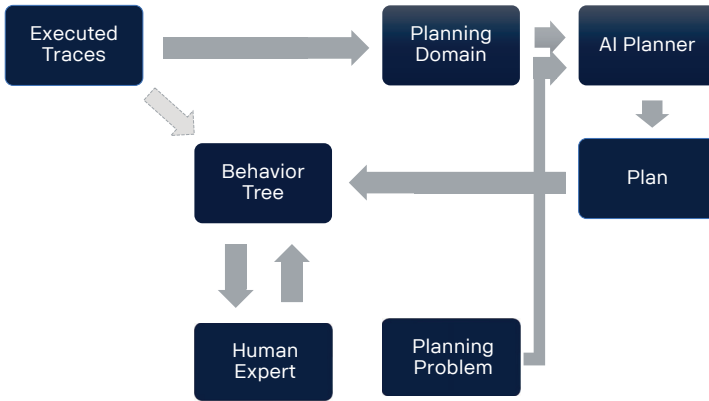


Figure 1.2: Overview of the final approach: symbolic planning models are learned directly from execution traces, while Behavior Trees (BTs) are used as an interface for human-centred interpretation, inspection, and refinement. This decoupled strategy preserves the interpretability benefits of BTs without relying on them as intermediate representations for model induction.

bridge between formal models and human understanding, supporting expert feedback and refinement.

Although expert feedback via BTs is not currently integrated into the domain learning loop, the visual interface enables validation and interpretability. Importantly, this feedback pertains to specific execution instances or plans rather than to the general domain model itself. As such, while this approach brings together domain learning and human-facing interfaces, it stops short of realizing a fully integrated human-in-the-loop system. Instead, it provides modular contributions that establish a foundation for future iterative frameworks, in which expert insight could directly influence model updates.

1.3.3 Validation

The methods proposed in this thesis have been validated through a combination of experiments targeting both individual components and selected parts of the overall learning pipeline. This multi-layered evaluation approach allows for isolating and analysing the behavior of key elements of the system, while also assessing how components interact.

Each stage of the pipeline has been assessed using a set of evaluation metrics specifically designed to reflect the assumptions, design goals, and intended function of that component.

In many cases, suitable evaluation metrics are not readily available in the existing literature. Where standard metrics are lacking, this thesis defines new ones. In doing so, it contributes toward the creation of more systematic evaluation practices for learning-based planning systems.

This validation strategy directly supports RQ4, which concerns the evaluation of the correctness and usefulness of learned action models

1.4 Contributions

The overall contribution of this thesis is to investigate how system behavior can be learned from execution traces in real-world autonomous transport systems, with a focus on enabling scalable and interpretable decision-making for fleet management. The unique aspect of this work is to identify representations and learning methods that are suitable not only for automated reasoning but also for human inspection, validation, and refinement. The six core contributions of this thesis are as follows:

- C_1 A method for learning Behavior Trees by integrating circuit theory with decision tree learning techniques.
This addresses RQ1, concerning the learning of behavior models, and it is discussed in Chapter 3.
- C_2 A method for learning behavioral models in the form of planning domains that can operate effectively in noisy environments, with a view toward real-world deployment.
This addresses RQ1, concerning model learning in real-world conditions, and it is discussed in Chapter 5.
- C_3 A framework for human-in-the-loop validation, debugging, and refinement of learned behavior models.
This addresses RQ2 and RQ3 regarding interpretability and human involvement, and it is discussed in Chapters 1 and 7.
- C_4 A principled evaluation framework for Behavior Trees by providing a structured guide comprising relevant properties, metrics, and design guidelines to support model development and analysis.
This addresses RQ4, regarding the evaluation of learned models, and it is discussed in Chapter 4.
- C_5 A systematic approach to evaluating learned planning domain models, encompassing both quantitative metrics and qualitative assessments.
This addresses RQ4, regarding model evaluation, and it is discussed in Chapter 6.

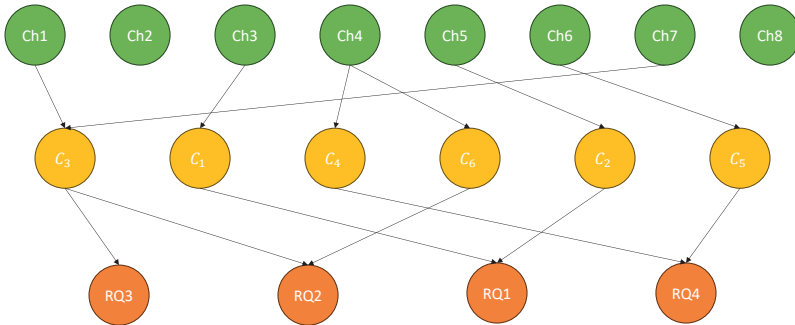


Figure 1.3: Interconnections among thesis chapters Ch (green), research contributions C (yellow), and research questions RQ (orange). Arrows represent direct relationships, where each element supports or informs the one it points to. This visual structure enables traceability across the thesis: from the structuring of the written document (chapters), through the underlying conceptual contributions, to the overarching research questions being addressed.

- C_6 An empirical study on the interpretability of Behavior Trees by conducting a user study.
This addresses [RQ2](#), regarding interpretability, and it is discussed in [Chapter 4](#).

1.5 Outline

This section provides an overview of the structure of the thesis and clarifies how each chapter contributes to the overall research goals. The relationship among research questions (RQ), contributions (C) and thesis chapters (Ch) is summarised in [Figure 1.3](#). This diagram offers a high-level overview of how the conceptual contributions are distributed across the thesis and how they collectively address the core research questions.

For example, [Chapter 4](#) supports Contributions 4 and 6, addressing Research Questions 4 and 2, respectively. This illustrates the logical progression from method developments and empirical analyses to the broader conceptual aims of the thesis.

The remaining chapters of the thesis are organised as follows:

Chapter 2 presents the necessary background to support the contributions of this thesis. It provides an overview of Boolean logic representations for cir-

cuit theory, decision trees, Behavior Trees (BTs), and automated planning. This foundational knowledge serves as the basis for the methods and frameworks proposed in the subsequent chapters.

Chapter 3 presents a novel method to learn BTs. It situates the proposed method within the existing literature and introduces a novel framework for constructing BTs by integrating circuit theory with decision tree learning. This chapter contributes to C_1 .

Chapter 4 tackles the challenge of evaluating BTs. It introduces a categorisation of key properties relevant to BT design and execution, along with corresponding metrics to measure them. Each property is rigorously defined, clarifying ambiguities found in prior literature. The chapter also provides design recommendations aimed at improving how BTs satisfy these properties. In addition, it presents a user study on the interpretability of BTs, contributing to both C_4 and C_6 . This work was developed in collaboration with David Caceres Domínguez and Marco Iannotta.

Chapter 5 presents a novel method for learning symbolic planning domains in noisy environments. It introduces a novel framework for acquiring action models suitable for real-world deployment, contributing to C_2 .

Chapter 6 addresses the evaluation of learned planning domain models. It situates the proposed method within the existing literature and proposes a systematic evaluation framework for assessing the correctness and utility of learned domains through a combination of formal metrics and qualitative considerations, contributing to C_5 .

Chapter 7 provides a comparative discussion between BTs and planning domains. It examines the challenges involved in translating BTs into planning domain representations and reflects on the limitations of planning domains from the perspective of human usability. This chapter contributes to the synthesis of the thesis and addresses C_3 .

Chapter 8 concludes the thesis by summarising the main contributions, discussing its technical limitations as well as its ethical and social implications, and outlining directions for future research.

1.6 List of Publications

The research conducted in this thesis has also resulted in a number of scientific publications. The following list includes those who have directly contributed

to the development of the thesis. Unless stated otherwise, I was the primary contributor to each publication, responsible for the main research ideas, the design and execution of experiments, and the majority of the writing.

- p_1 *Gugliermo, S.*, Learning planning domains for intelligent transport systems. In Proceedings of the 35th Swedish Artificial Intelligence Society (SAIS'23) Annual Workshop, 2023. [63]
 This extended abstract introduces the scientific and industrial motivation behind the thesis. It outlines the overarching problem of automating planning domain acquisition for autonomous transport systems and presents the initial vision and research direction.
Parts of Chapter 1
- p_2 *Gugliermo, S.*, Schaffernicht, E., Koniaris, C., Saffiotti, A. Extracting planning domains from execution traces: a progress report. In Proceedings of the ICAPS 2023 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS), 2023. [66]
 This workshop paper presents the first learning architecture explored in the thesis, where Behavior Trees (BTs) are used as intermediate structures between execution traces and symbolic planning models. It highlights the conceptual motivations and the challenges of this approach.
Parts of Chapter 1
- p_3 *Gugliermo, S.*, Schaffernicht, E., Koniaris, C., Pecora, F. Learning behavior trees from planning experts using decision tree and logic factorization. In IEEE Robotics and Automation Letters (RA-L), 8(6):3534-3541, 2023. [65]
 This article proposes a method to learn BTs from execution traces using decision tree induction and logic factorization. It focuses on capturing expert behavior patterns in an interpretable and structured format suitable for inspection and reuse.
Chapter 3
- p_4 *Gugliermo, S.*, Domínguez, D. C., Iannotta, M., Stoyanov, T., Schaffernicht, E. Evaluating behavior trees. In Robotics and Autonomous Systems, 178, 104714, 2024. [64]
 This work systematically defines and analyzes key properties for evaluating Behavior Trees, such as modularity, robustness, and reactivity. Each property is formally characterized, linked to relevant metrics, and illustrated through practical examples. The paper also provides design guidelines for improving BTs along these dimensions. This is the result of collaborative work conducted with David Caceres Domínguez and Marco Iannotta. David, Marco, and I contributed

equally to the conceptualization of the evaluation framework, the formalization of property definitions, and the implementation of the associated metrics.

Chapter 4

- p*₅ *Gugliermo, S., Köckemann U., Schaffernicht, E., Saffiotti.* Learning Lifted Action Models for Planning Domain Acquisition in Noisy Environments. [under review]

This submission extends the scope of planning model acquisition to settings with noise and partial observability. It proposes a method for learning lifted representations that generalize across entities and contexts, enabling model reuse and transferability. In addition, it addresses the challenge of evaluating the quality of the learned planning representations.

Chapters 5 and 6

- p*₆ *Gugliermo, S., Domínguez, D. C., Schaffernicht, E., Stoyanov, T., Renoux, J.* BT User Study. [under review]

This study investigates how practitioners interpret and evaluate Behavior Trees. This is the result of collaborative work conducted with David Cáceres Domínguez. David and I contributed equally to the conceptualization of the work, the design and execution of the user study, data analysis, and the articulation of results and insights regarding BT interpretability.

Chapter 4

Chapter 2

Background

In this chapter, an overview of key concepts that are fundamental to the remainder of this thesis is provided. These concepts will serve as the groundwork for the following chapters, where specialised methods and case studies will be explored in detail.

2.1 Boolean Logic Representations

Boolean logic is the main formalism underlying the method presented in Chapter 3. It is a fundamental mathematical framework used to represent and manipulate logical expressions. It serves as the foundation for digital circuits, computer science, and artificial intelligence. Boolean logic operates on binary values (true and false, or 1 and 0) and uses logical operations to process and evaluate expressions efficiently. In this thesis, Boolean logic is specifically discussed because it provides the necessary formalism for representing and manipulating preconditions within action models, a core topic of investigation in the subsequent chapters. Understanding Boolean logic is crucial for expressing, analysing, and optimising the logical rules and constraints that govern autonomous systems.

Moreover, Boolean logic supports advanced techniques such as logic synthesis, minimisation, and optimisation, which are vital for scalable reasoning. Representational strategies such as algebraic logical trees, factored forms, and canonical normal forms provide the structural basis for systematic manipulation of Boolean functions, facilitating both theoretical analysis and practical implementation. In this thesis, the definitions and formal structures introduced by Wang [170] are adopted as the primary formal basis.

2.1.1 Fundamental Concepts

To facilitate the discussion, several key definitions used in Boolean logic representation are introduced:

Definition 2.1.1 (Well-formed-formula). A well-formed formula (*wff*) in Boolean logic is an expression, e.g., $a \cdot b + \neg c$, where the symbols \cdot , $+$, \neg denote conjunction (AND), disjunction (OR), and negation (NOT), respectively.

Definition 2.1.2 (Literal). A *literal* is a variable or its negation.

Definition 2.1.3 (Cube). A *cube* is a set C of literals such that $x \in C$ implies $\neg x \notin C$ (e.g., $\{a, b, \neg c\}$ is a cube), and represents the conjunction of its literals.

Definition 2.1.4 (Expression). An *expression* is a set f of cubes.

Definition 2.1.5 (Clause). A *clause* is either a single literal or the disjunction of two or more literals, e.g., $a + \neg b + c$.

2.1.2 Algebraic Trees and Factorization

An algebraic tree is a hierarchical representation of a Boolean function, where each node corresponds to an operation (AND, OR, NOT) and the leaves represent literals or constants. Algebraic trees provide a structured approach to handling Boolean expressions, enabling efficient manipulation, transformation, and simplification.

Definition 2.1.6 (Algebraic tree). An *algebraic tree* is a rooted tree where:

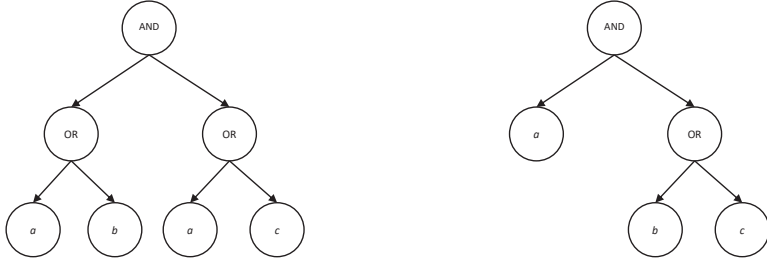
- internal nodes represent Boolean operators (AND, OR, NOT).
- Leaf nodes correspond to Boolean variables or constants (0 or 1).
- Each subtree defines a valid Boolean sub-expression.

For example, consider the Boolean function $a \cdot b + a \cdot c$. Its corresponding algebraic tree representation is depicted in Figure 2.1a.

Logic factorization is the process of deriving a factored form from a given Boolean expression. A factored form is a parenthesized algebraic expression that represents the logical function in a more compact and structured manner. Using factorization, the Boolean expression $a \cdot b + a \cdot c$ can be rewritten as $a \cdot (b + c)$, simplifying the logic representation (Figure 2.1b). Factorization relies on Boolean algebra rules.

2.1.3 Boolean Algebra

Boolean algebra provides a systematic framework for defining and manipulating logical expressions. In Boolean algebra, any *wff* can be stated in standard forms to facilitate computation and analysis:



(a) Expanded algebraic tree representation.

(b) Factorized algebraic tree representation.

Figure 2.1: Transformation of an algebraic tree through factorization. The initial tree (left) directly represents the expression $a \cdot b + a \cdot c$, while the transformed tree (right) illustrates its factorized form $a \cdot (b + c)$.

- Disjunctive Normal Form (DNF): A sum of products representation
- Conjunctive Normal Form (CNF): A product of sums representation

In this thesis, *Horn clauses* are leveraged to express logical rules in CNF. Horn clauses are a subset of Boolean logic expressions, particularly useful in logic programming and automated reasoning.

Definition 2.1.7 (Horn clauses). A Horn clause is a disjunction of literals with at most one positive (non-negated) literal

For example, the expression $\neg a + \neg b + \neg c + d$ is a Horn clause. Horn clauses express rules and can be equivalently written in the form of implications, e.g., $a \cdot b \cdot c \implies d$, which can be interpreted as

if a and b and c all hold, then also d holds.

This describes the structure of a rule in which a , b and c are the conditions to be met for an action d to be applied.

Boolean algebra consists of several fundamental operations:

- AND (Conjunction): $a \cdot b$ is true only if both a and b are true.
- OR (Disjunction): $a + b$ is true if at least one of a or b is true.
- NOT (Negation): $\neg a$ inverts the truth value of a .

In this thesis, apart from these operations, a more complex operation will be leveraged to further manipulate Boolean expressions efficiently.

Definition 2.1.8 (Division). A *division* in Boolean algebra is an operation which, given expressions f and p , finds expressions q and r such that $f = p \cdot q + r$.

Note that such a division operation is not unique. Indeed, the resulting q and r may be dependent upon the particular representation of f and p . In this thesis, Boolean expressions are treated as polynomials and therefore the conjunction $p \cdot q$ is seen as an algebraic product.

Definition 2.1.9 (Cube-free expression). An expression is *cube-free* if no cube divides the expression evenly.

For example, $a \cdot b + c$ is cube-free, while $a \cdot b + a \cdot c$ is not cube-free because a is a common factor.

2.2 Decision Trees

Decision trees will be used as an interpretable intermediate representation in the process of learning models of system behavior. They are a fundamental and widely used model in machine learning and data analysis [115], assisting in decision-making by structuring data into a hierarchical, tree-like format. They recursively partition input data based on feature values, facilitating predictions for classification and regression tasks [17]. Each internal node represents a decision based on a specific feature, edges denote possible outcomes, and leaf nodes correspond to predicted values or classes (Figure 2.2).

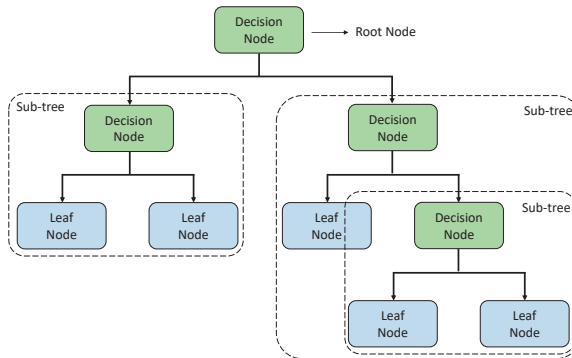


Figure 2.2: Example of a decision tree.

Due to their simplicity, interpretability, and efficiency, decision trees are applied across various domains, including finance, healthcare, and marketing,

where they handle both categorical and numerical data effectively [67]. In the context of this thesis, decision trees are introduced because their structure is directly learned from data and subsequently used as a scaffold for constructing Behavior Trees (see Section 2.3). Understanding decision trees is, therefore, a necessary prerequisite for comprehending the methods for behaviour modelling developed in later chapters.

To illustrate the functionality of decision trees, consider a simple classification example in which the goal is to predict whether a person will purchase a product based on two features: Age and Income. The target variable represents the purchasing decision (Buy = Yes or No). The corresponding dataset is presented in Table 2.1. Based on this data, the decision tree may be structured as shown in Figure 2.3. Initially, the tree splits the data based on the Income feature. If Income is Low, the decision follows the left branch of the tree. The next decision is then based on Age, further refining the classification process.

Age	Income	Buy
25	Low	No
31	High	Yes
34	Low	No
35	Low	Yes
45	High	Yes
50	Low	Yes
55	High	Yes

Table 2.1: Example dataset for decision tree learning.

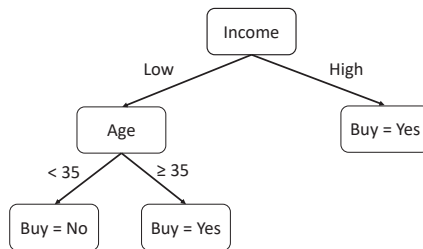


Figure 2.3: Decision tree associated with dataset in Table 2.1.

2.2.1 Learning of Decision Trees

The primary objective of learning decision trees from data is to construct a model that accurately predicts the target variable by systematically partitioning the feature space. At each step, the algorithm selects the feature and corresponding threshold that best separates the data, creating a tree structure where each internal node represents a decision based on attribute values. This process continues until a predefined stopping condition is met, such as reaching a maximum tree depth, having too few samples in a node, or when further splits do not significantly improve predictive performance.

The decision tree learning process is inherently greedy, which implies that decisions are made based on local optimizations at each node without considering the overall tree structure. Different decision tree algorithms employ different criteria to determine the best split at each node. The most common criteria are based on:

- **Gini Impurity:** Measures how often a randomly chosen instance would be incorrectly classified if labeled according to the class distribution in a given node. It is commonly used in classification trees like CART.
- **Information Entropy:** Measures the degree of disorder or uncertainty in the dataset. It is used in decision trees like ID3, C4.5, and C5.0 [136] to determine which feature provides the most information about the target variable by reducing entropy.

Despite their tendency to overfit, they offer a powerful means to model complex relationships in data, especially when combined with other techniques like pruning, ensemble methods, or boosting.

C5.0 Algorithm

In this thesis, C5.0 [95, 96] is used for decision tree learning. C5.0 evaluates potential splits in the dataset based on two key measures:

- **Entropy:** Measures the impurity or randomness of a dataset. A node with high entropy indicates that the data is not well separated and requires further splitting. It is defined as:

$$Entropy(S) = -\sum p(i) \cdot \log_2(p(i))$$

where S represent the dataset and $p(i)$ is the percentage of data points belonging to the class i .

- **Information Gain:** Determines the effectiveness of a feature in reducing entropy after a split. A higher information gain means the feature is more useful for classification. It is calculated as:

$$Gain(S, A) = Entropy(S) - \sum \left(\frac{|S_v|}{|S|} \right) \cdot Entropy(S_v)$$

where S represent the dataset, A is the feature used for splitting, S_v is a subset of S corresponding to an attribute value, $|S_v|$ is the number of data points in S_v and S is the number of data points in S .

C5.0 improves upon information gain by introducing the **Gain Ratio**, which normalizes information gain to prevent bias toward attributes with many distinct values:

$$\text{Gain Ratio}(A) = \frac{\text{Information Gain}(A)}{\text{Split Information}(A)}$$

where $\text{Split Information}(A)$ measures the intrinsic uncertainty in an attribute's values, and it is computed as the distribution's entropy for attribute A 's values. This normalization ensures a more balanced selection of splitting features.

Algorithm 1 shows a simplified pseudocode representation of the C5.0 decision tree algorithm. The decision tree is built recursively by selecting the attribute with the highest Gain Ratio, partitioning the dataset, and repeating the process for each subset.

The dataset is recursively divided according to the chosen attribute that yields the maximum information gain (lines 7-11). At each step, the algorithm evaluates the remaining data and determines whether further splits are necessary. If all instances in a subset belong to the same class, the algorithm terminates the partitioning process and assigns a leaf node labeled with that class (lines 2-3). If no more attributes are available for further splits, the algorithm assigns a majority class label to the subset (lines 4-5). When stopping conditions are met, the algorithm finalizes the tree structure, ensuring that it effectively generalizes to new data without excessive complexity.

Algorithm 1 C5.0 decision tree algorithm

```

1: function C5.0(Data, Attributes)
2:   if all examples in Data belong to the same class then
3:     return a leaf node with the class label
4:   else if Attributes is empty then
5:     return a leaf node with the majority class label in Data
6:   else
7:     Select the best attribute  $A$  using Information Gain
8:     Create a decision node for  $A$ 
9:     for each value  $v$  of  $A$  do
10:      Create a branch for  $v$ 
11:      Recursively apply C5.0 to the subset of Data where  $A = v$ 
12:   return the decision tree

```

Once the tree is fully constructed, it can be represented either as a *decision tree* or as a *set of rules*, which offer a more compact and human-readable

format. In cases where multiple rules apply to the same instance (i.e., all conditions of more than one rule are met, leading to an implicit conflict), C5.0 resolves conflicts by selecting the rule with the highest confidence. If multiple rules remain applicable, the algorithm employs a weighted voting mechanism, where each rule contributes to the final classification based on its confidence level. If no rule matches an instance, a *default class* is assigned to ensure full coverage of all possible cases.

Once the tree is fully grown, C5.0 applies pruning, which eliminates superfluous branches that are less important for overall generalization and more for fitting the training set. C5.0 uses a cost-complexity pruning strategy to strike a compromise between the decision tree's mistake rate and complexity. It computes the minimal error reduction needed to keep a branch using a confidence factor. If a branch does not meet this requirement, it is pruned from the tree. By pruning less informative branches, C5.0 produces a more compact and interpretable model, reducing overfitting while maintaining strong predictive performance

2.3 Behavior Trees

Behavior Trees (BTs) were the first representation used in this work to learn system behavior, as described in Chapter 3. BTs are used to describe a policy of an autonomous agent and are widely applied in robotics, video games, and AI-driven systems due to their modularity, reactivity, and transparency. In this thesis, BTs are introduced because their hierarchical and interpretable structure provides a suitable target for transforming learned decision trees into behavior models. This transformation enables human-in-the-loop validation and aligns with the overarching goal of developing transparent and modular representations for autonomous system behaviour. Formally, a BT is a directed acyclic graph (DAG) in which the internal nodes, known as control flow nodes, determine execution logic, while leaf nodes, referred to as execution nodes, perform specific actions. The root node has no parent, whereas all other nodes have exactly one. Control flow nodes have at least one child, arranged graphically below the parent node [29], as illustrated in Figure 2.4.

The execution of a BT begins from the root node, which generates signals called *ticks* at a given frequency. These ticks propagate through the tree, reaching control flow and execution nodes in a depth-first order. A node is executed only when it receives a tick and returns one of the following statuses to its parent:

- *Success*: The node successfully completed its task.
- *Failure*: The node could not complete its task.
- *Running*: The node is still executing its task.

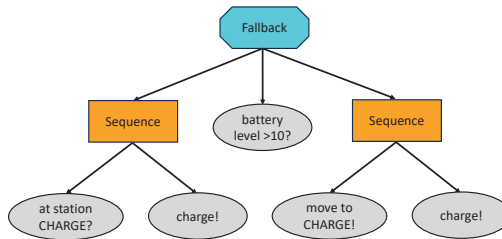


Figure 2.4: Example of a Behavior Tree for the battery management of an autonomous agent. If the robot is at a charging station, it charges its battery. If it is not at the charging station but the battery level is above 10%, the condition node returns *Success*, and the robot continues its current task. If the battery level drops below 10%, the robot navigates to the charging station to recharge.

2.3.1 Classical Behavior Tree

This work adopts the standard formulation of BTs, which defines a fixed set of control flow and execution nodes with deterministic tick-based semantics. Control flow nodes determine the execution order and the propagation of ticks according to predefined logic.

The standard control flow nodes are:

- **Sequence:** Processes ticks by routing them to its children sequentially from left to right, following a depth-first order. It returns *Success* only if all its children return *Success*. If any child returns *Running* or *Failure*, the *Sequence* node halts further propagation of ticks to subsequent children. This behaviour is conceptually similar to the logical AND operator and is visually represented in this thesis as an orange rectangle.
- **Fallback:** Processes ticks by routing them to its children sequentially from left to right, following a depth-first order. It returns a status of *Success* as soon as any child returns *Success*. If a child returns *Success* or *Running*, the *Fallback* node stops propagating ticks to subsequent children. Only when all children return *Failure* it returns *Failure*. This behaviour is analogous to the logical OR operator and is visually represented in this thesis as a blue octagon.

- **Parallel:** Routes ticks to its children simultaneously. Its behaviour is determined by a user-defined parameter M : the node returns *Success* if at least M out of the N children return *Success*. conversely, it returns *Failure* if more than $N - M + 1$ children return *Failure*. If neither condition is satisfied, the node returns *Running*. This behaviour can be likened to the logical operator OR if $M = 1$, and in this thesis, is graphically represented as a yellow parallelogram.

It is important to note that the analogies between control flow nodes and logical operators relate to the execution of logical expressions, where the order of evaluation proceeds from left to right.

Execution nodes represent the leaves of the BT and are responsible for performing specific tasks or evaluating conditions. The standard execution nodes are:

- **Action:** Represents the system's capabilities and corresponds to physical actions or commands that the agent can execute. When an action node receives ticks, it initiates the execution of its associated command. It returns *Success* if the action is successfully completed, *Failure* if it fails, and *Running* while the action is still in process. In this thesis, action nodes are represented as grey ovals, and their names are followed by an exclamation mark ("!") to indicate their action-oriented nature.
- **Condition:** Evaluates logical conditions. Upon receiving ticks, a condition node checks whether a specific condition or proposition holds. It returns *Success* if the condition is satisfied and *Failure* if it is not. Note that a Condition node never returns a status of *Running*. In this thesis, condition nodes are visually represented as grey ovals, and their names are followed by a question mark ("?") to indicate their evaluative nature.

BTs are particularly effective for representing sequential, conditional, and parallel behaviors. Sequence nodes enforce a structured execution order, ensuring that tasks are only executed if all preceding ones succeed. This makes them ideal for modeling workflows that require step-by-step completion. Fallback nodes, on the other hand, enable the system to attempt alternative strategies, ensuring robust decision-making in uncertain environments.

A significant advantage of BTs is their reactivity, enabled by the periodic ticking mechanism. This allows the tree to dynamically adapt to changes in the environment or system state, ensuring that critical conditions (such as safety checks or resource availability) are evaluated before executing actions. The hierarchical arrangement of tasks naturally prioritizes essential operations, with higher-priority tasks placed on the left and evaluated first.

Additionally, BTs exhibit modularity and transparency, making them highly scalable and interpretable. Modular subtrees allow for easy extension and reuse in more complex behavior structures, while the hierarchical execution logic

ensures that decision-making remains clear and interpretable. This structure simplifies debugging and enhances the explainability of autonomous agent behaviors.

2.3.2 Non Classical Behavior Trees

While this thesis adopts the classical formulation of Behavior Trees (BTs), it is important to acknowledge the emergence of several non-classical variants in the literature that extend the expressive power and adaptability of BTs. Although such variants offer increased flexibility, they often compromise the modularity and interpretability that make classical BTs particularly suitable for symbolic reasoning and human-in-the-loop validation.

Dynamic Behavior Trees (DBTs) [12] refer to BT variants that support runtime modifications to their structure, enabling agents to adapt their behavior dynamically in response to changing environments or goals. Unlike classical BTs, which are static once constructed, DBTs can insert, remove, or reconfigure subtrees during execution. This capability allows for increased flexibility, enabling agents to handle unforeseen situations or evolving objectives without requiring the entire behavior model to be redesigned. However, this adaptability comes at a cost: runtime changes can reduce predictability and complicate formal analysis and verification. Therefore, classical BTs are easier to analyze and validate due to their fixed structure, while dynamic BTs offer greater flexibility at the expense of transparency and determinism.

Stochastic Behavior Trees (SBTs) [24] are an extension introduced to tackle a specific design challenge of classical BTs: the execution order of actions or conditions in a BT is fixed and must be decided a priori by the designer. SBTs introduce probabilistic or randomized selection mechanisms into control nodes, allowing the tree to explore multiple execution paths over time. This nondeterminism can improve behavioral robustness and long-term performance by mitigating the impact of suboptimal action sequences. However, the trade-off is reduced predictability: SBTs may yield different outcomes under identical conditions, which can complicate debugging and reasoning about behavior. Thus, while SBTs enhance expressiveness and adaptability, they sacrifice the deterministic execution and ease of traceability characteristic of classical BTs.

Conditional Behavior Trees (CBTs) introduce preconditions and post-conditions into the BT framework to improve integration with goal-directed behavior [54]. However, these extensions remain tightly coupled to specific execution contexts, and the conditions they encode are typically hand-crafted or bound to low-level system states. Consequently, while CBTs enhance the

structuring of reactive behavior, they do not directly support the kind of abstract, domain-level reasoning required for automated planning. The reliance on explicit, implementation-specific semantics limits their utility in inferring generalisable planning models from observed behavior, a central objective of this thesis.

When comparing classical BTs to these non-classical extensions, a consistent pattern emerges: while dynamic, stochastic, and conditional BTs increase the expressiveness and adaptability of the framework, they do so at the cost of added complexity and diminished transparency. Classical BTs remain attractive for tasks where modularity, interpretability, and human-in-the-loop validation are paramount, making them a compelling foundation for symbolic reasoning and explainable planning.

2.4 AI Planning

As will be discussed in Chapter 5, the second system behavior representation adopted in this work is PDDL, the standard formalism in AI planning. Task-level or high-level planning, also referred to as symbolic Automated Planning (AI Planning), is a branch of Artificial Intelligence that focuses on automatically generating action sequences to achieve specific goals within a system [51].

Definition 2.4.1 (Planning Problem). A planning problem can be described as a tuple $\Pi = \langle P, S_0, S_g, A \rangle$, where P is a set of predicates used to describe states, S_0 is the initial state, S_g is the goal condition, and A is the set of available actions in the system \mathcal{S} .

The solution to a planning problem is a plan, which is formally defined as follows:

Definition 2.4.2 (Plan). A plan π is a sequence of actions $\langle a_0, a_1, \dots, a_n \rangle$ such that if these actions are applied in order from the initial state s_0 , the resulting sequence of state transitions $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$, satisfying any relevant constraints and such that $s_n \in S_g$

A plan can be viewed as a collection of traces, where each trace captures the state transitions occurring at each step of the plan.

Definition 2.4.3 (Trace). A trace t is a set of n transitions:

$$t = \{\langle s_i, a_i, s'_i \rangle\}_{i=1}^n$$

where each transition $\langle s_i, a_i, s'_i \rangle$ consists of:

- s_i - The state before executing action a_i .
- a_i - The action applied in state s_i .

- s'_i - The resulting state after executing a_i , computed as:

$$s'_i = s_i \setminus \text{eff}^-(a_i) \cup \text{eff}^+(a_i)$$

where $\text{eff}^+(a_i)$ and $\text{eff}^-(a_i)$ are defined formally in Definition 2.4.9.

Given a plan $\pi = (a_1, \dots, a_n)$, the *plan trace* of π is a trace:

$$t_\pi = \{\langle s_{i-1}, a_i, s_i \rangle\}_{i=1}^n$$

where the transitions are generated by executing π from an initial state s_0 to reach the goal state.

Similarly, a set T of m traces can be defined as the union of individual traces:

$$T = \bigcup_{i=1}^m t_i$$

2.4.1 Classical Planning

This thesis primarily focuses on classical planning.

Definition 2.4.4 (Classical Planning). Classical planning involves the offline generation of action sequences for a deterministic, static, finite system, with complete knowledge, attainment goals, and implicit time.

Therefore, classical planning operates under a set of assumptions [51]

1. Finite state space: The system \mathcal{S} has a finite set of states
2. Fully observable state: The system \mathcal{S} is fully observable - the planner has complete knowledge of the world's state \mathcal{S} .
3. Deterministic actions: Actions have deterministic effects - each action in a given state leads to at most one resulting state
4. Static world: The system \mathcal{S} is static - no exogenous events occur. The state only changes when the planner's actions occur. Hence, the system remains static unless controlled transitions take place
5. Restricted goals: The planner handles only restricted goals that are unspecified as an explicit goal state s_g or set of goal states S_g ; the objective is any sequence of state transitions that ends at one of the goal states. Extended goals, such as states to be avoided and constraints on state trajectories or utility functions, are not handled under this assumption.
6. Sequential plans: Plans are sequences of actions with a linear order (no concurrency)

7. Implicit time: Actions are instantaneous (no duration), and time is not modeled explicitly
8. Offline planning: Planning is done in an open-loop manner, before execution - the planner does not receive feedback or adapt to changes during plan execution.

Formally, a classical planning problem is defined as follows

Definition 2.4.5 (Classical Planning Problem). A classical planning problem is a tuple $\mathcal{P} = \langle \Sigma, s_0, G \rangle$, where Σ is a transition system capturing the dynamics of the environment, s_0 is the initial state, and S_g is a specification of the goal condition.

Definition 2.4.6 (Transition System). The transition system Σ is a tuple $\Sigma = (S, A, \gamma)$, where S is the (finite) set of all possible states, A is the set of actions (also called operators), and $\gamma : S \times A \rightarrow S$ is a deterministic state-transition function that defines the next state $\gamma(s, a) = s'$ resulting from applying action a in state s .

In classical planning, actions are deterministic; hence, for every (s, a) , the transition function γ produces a single successor state rather than a set of possible outcomes.

To solve a planning problem, one must formally describe the *planning domain* (the general environment and action dynamics) and the specific *problem instance* (the initial state and goal for a particular scenario). When describing these by using the common planning languages, like STRIPS and PDDL, people separate these descriptions into a domain model (defining the actions and predicates common to all problems in that domain) and a problem specification (defining the particular objects, initial state, and goal for the instance at hand). Together, the domain and problem instance description provide all the information needed for a planner to search for a solution plan. Most planners solve planning problems by employing some form of state-space search. In this model, the initial state serves as the root, and actions are applied to generate successor states, resulting in a directed graph of possible states. However, in practice, many search algorithms enforce a tree-like exploration by keeping track of visited states to avoid redundant expansions. The task of the planner is then to search this space and identify a path that leads from the initial state to a state satisfying the goal, ideally with minimal cost or depth. Therefore, the efficiency of a planner is largely determined by search algorithms, heuristic strategies, and the representations of planning domains.

Planning Domain Definition

A planning domain describes the general environment, the types of objects and predicates, and the actions available for all problems within that domain.

It provides a structured representation of how the system transitions from one state to another, enabling the planner to reason about possible future states and generate appropriate sequences of actions to achieve the desired goal.

A well-defined planning domain is essential for ensuring the applicability, correctness, and efficiency of the plans generated by the AI planner. It serves as an abstraction of the real-world system, encapsulating the knowledge required for planning, without specifying any particular planning problem instance. By capturing the actions, their preconditions, and effects, the domain provides a formalized framework that supports reasoning and planning in dynamic environments.

Definition 2.4.7 (Planning Domain). A planning domain D is a triple $\langle S, A, H \rangle$, where:

- S is a set of state variables (predicates) that describe the state of the environment.
- A is a set of action (operator) names, each associated with an arity.
- H is a function mapping each action $a \in A$ to its operator specification, i.e., a pair $\langle \text{pre}(a), \text{eff}(a) \rangle$ defining its preconditions and effects.

Where arity is defined as follows:

Definition 2.4.8 (Arity). The *arity* of an action or state variable is the number of arguments it requires. Formally, given an action or state variable p , its arity is denoted as $\text{arity}(p)$ and defines the number of terms that must be assigned when instantiating p .

State variables and actions of arity n are called n -ary state variables and n -ary actions.

The key elements of a planning domain include:

- **Types:** A classification of objects into categories that define their roles within the domain. For example, in an elevator domain, we may define types such as elevator, person, and floor.
- **Objects:** The set of entities that exist in the world. In a typed domain, each object belongs to a specific type (e.g., floor_1 is of type *floor*).
- **Predicates:** Boolean-valued functions that describe properties or relationships between objects. Predicates define state descriptions and action conditions. For instance, $\text{At}(\text{elevator}, \text{floor})$ denotes that an elevator is at a certain floor.
- **Actions:** Formally defined as,

Definition 2.4.9 (Action). An action is a tuple:

$$\langle \text{par}(a), \text{pre}^+(a), \text{pre}^-(a), \text{eff}^+(a), \text{eff}^-(a) \rangle$$

where:

- $\text{par}(a)$ is a tuple of variables representing the parameters of the action.
- $\text{pre}^+(a)$ is the set of positive preconditions (i.e., state variables that must be true for the action to be applicable).
- $\text{pre}^-(a)$ is the set of negative preconditions (i.e., state variables that must be false for the action to be applicable).
- $\text{eff}^+(a)$ is the set of positive effects (i.e., state variables that become true after executing the action).
- $\text{eff}^-(a)$ is the set of negative effects (i.e., state variables that become false after executing the action).

A ground action is a fully instantiated version of an action where all parameters are replaced with specific constants from the domain. Formally, it is defined as follows:

Definition 2.4.10 (Ground Action). Given an n -ary action $a \in A$ and a set of constants c_1, \dots, c_n , the *grounded action* $\hat{a} = \text{op}(c_1, \dots, c_n)$ is a 4-tuple:

$$\langle \text{pre}^+(\hat{a}), \text{pre}^-(\hat{a}), \text{eff}^+(\hat{a}), \text{eff}^-(\hat{a}) \rangle$$

where each of $\text{pre}^+(\hat{a}), \text{pre}^-(\hat{a}), \text{eff}^+(\hat{a}), \text{eff}^-(\hat{a})$ is obtained by substituting every occurrence of a parameter $x_i \in \text{par}(a)$ with the constant c_i in the corresponding position.

This notion of grounding extends naturally to other components of a planning domain. A ground precondition or ground effect is a set of predicates in which all parameters are instantiated with specific constants. A grounded domain is a planning domain in which all actions have been fully instantiated with all possible combinations of constants from the object set, yielding a finite set of ground actions.

In contrast, lifted actions, lifted preconditions, lifted effects, and lifted domains retain variables and refer to general object types rather than specific instances. These lifted representations are more compact and general, allowing planners to reason symbolically and instantiate actions only as needed. The term lifted reflects the abstraction from concrete instances to reusable, parameterized templates.

A planning domain is typically defined separately from any specific problem instance so that the same domain can be reused across multiple problems with different initial states and goals.

Planning domains are often represented in a standard language that provides a structured formalism for defining predicates, actions, and their effects, allowing planners to operate in a standardized way, making it possible to benchmark different planning algorithms across various domains.

The Planning Domain Definition Language (PDDL)[113] is a formal language designed to represent planning problems. Initially introduced for the International Planning Competition (IPC), PDDL has become the standard in AI planning research. Over time, it has evolved through several versions, each adding features to enhance its expressiveness and applicability.

- PDDL 1.2 [113]: The first version, based on STRIPS and ADL, defined classical planning problems.
- PDDL 2.1 [44]: Introduced numeric fluents (for resources), durative actions (for time-dependent effects), and continuous change.
- PDDL 2.2 [41]: Added derived predicates and timed initial literals.
- PDDL 3.0 [50]: Introduced soft goals (preferences) and trajectory constraints, allowing for optimization based on preferred goals.

PDDL's evolution has made it increasingly capable of modelling complex planning scenarios, making it the most widely used language for defining planning problems.

Planning Problem Definition

A planning problem is an instance within a given planning domain that specifies the concrete details of a specific task to be solved. It consists of:

- The Initial State (S_0): The fully specified starting configuration of the world. The initial state is often described as a set of logic atoms considered true in that state. In an elevator domain, this could be: $\text{At}(\text{elevator1, floor4}) \wedge \text{At}(\text{person1, floor2})$
- The Goal Condition (S_G): A set of logical atoms or logical conditions that define when the problem is considered solved. Unlike the initial state, the goal is usually a partial specification, meaning that not all facts need to be specified, only those that must hold for the problem to be solved. In an elevator domain, this could be: $\text{At}(\text{person1, floor1})$.

Given a well-defined planning domain and a specific problem instance, a planner searches for a sequence of actions that transforms the initial state into a state that satisfies the goal condition.

2.4.2 Non Classical Planning

While classical planning offers formal clarity and algorithmic tractability, it is limited in its ability to model more realistic scenarios where uncertainty, hierarchy, or temporal constraints are present. Several non-classical approaches have been developed to address these limitations.

Hierarchical Task Network (HTN) Planning [177] models complex tasks as hierarchically structured subtasks, where a high-level task is decomposed into simpler ones until reaching primitive actions. HTN planning is particularly effective for structured problem domains where expert knowledge can guide the decomposition process. It requires domain-specific knowledge in the form of decompositions, which cannot easily be inferred from observational data alone. To support such hierarchical modelling, HTN-specific planning languages such as HDDL [72], have been developed that extend classical representations with decomposition operators. However, the primitive actions in HTN models (the lowest-level units that are ultimately executed) must be defined using the same formal structures as classical planning. Thus, classical planning serves as a foundational layer upon which HTN methods are built. This further motivates the focus of this thesis on classical models, as they constitute the basis for any high-level reasoning or abstraction mechanism that ultimately requires execution-level semantics.

Temporal Planning [51] extends classical planning to include time constraints. Actions have durations, and plans need to satisfy time constraints while ensuring task feasibility. Temporal extensions introduce significant representational and computational complexity. Plans are no longer simple action sequences but must satisfy a range of temporal constraints, often requiring explicit handling of scheduling, resource usage, or concurrency. To accommodate such features, temporal extensions of PDDL, such as PDDL 2.1 [44], have been developed. In the context of this thesis, such temporal expressiveness is not required and would introduce unnecessary complexity. Classical planning, by contrast, provides a compact, discrete, and semantically transparent framework that is better aligned with the thesis’s focus on symbolic learning, causal reasoning, and interpretability.

Probabilistic and Stochastic Planning [98, 88] accounts for uncertainty in action outcomes using probabilistic models such as Markov Decision Processes (MDPs) and Partially Observable Markov Decision Processes (POMDPs). These approaches enable planning in environments where the effects of actions are not deterministic. This method, while capable of handling uncertainty, tends to obscure causal structure and hinder symbolic abstraction. Its reliance on probabilistic inference complicates model interpretability and traceability,

both of which are essential for symbolic model validation and human-in-the-loop workflows. Languages such as PPDDL (Probabilistic PDDL) [18] and RDDDL (Relational Dynamic Influence Diagram Language) [148] have been developed to express stochastic and decision-theoretic planning problems. Despite their expressive power, these languages rely on numerical parameters and statistical inference, which can hinder symbolic abstraction and human-centred validation. In contrast, classical planning offers transparent and deterministic semantics that better support causal reasoning and systematic model evaluation.

Conditional planning [55] addresses the challenge of incomplete information and the need for plans that include contingent branches depending on the state of the environment during execution. Unlike classical planning, which assumes full observability and generates linear plans, conditional planning produces branching plans that specify actions to take under different conditions. These plans are often constructed using languages that support sensing and conditional logic, such as ADL [130]. Managing branches and sensing actions complicates both the learning process and the interpretability of the resulting models. Since this thesis focuses on learning action models from fully observed execution traces in deterministic settings, conditional planning lies outside the scope of the present work. Nonetheless, it represents an important direction for future extensions.

Although non-classical planning paradigms offer valuable extensions to handle uncertainty, hierarchy, and time, this thesis deliberately focuses on classical planning. Classical planning models offer deterministic semantics, symbolic transparency, and structural simplicity. These properties are critical for the thesis’s aims: to learn symbolic action models from execution traces and to support interpretability and human validation. Furthermore, many non-classical approaches retain classical planning as a foundational layer. HTN planning depends on classical primitives, probabilistic models often extend classical syntax, and temporal planning builds on PDDL. By focusing on classical planning, this thesis establishes a robust and interpretable foundation upon which more expressive paradigms can later be developed.

Chapter 3

Learning Behavior Trees

A fundamental goal in learning action models is not only to derive accurate representations of system behavior but also to represent them in a structured and interpretable manner. The ability to visualize learned action models can significantly enhance understanding, debugging, and deployment in real-world applications.

Behavior Trees (BTs) provide a powerful framework for structuring and visualizing learned action models. Their intuitive, human-readable nature makes them particularly useful for explicitly representing decision sequences, conditional logic, and action execution flow.

Unlike classical planning representations that rely on symbolic preconditions and effects, such as STRIPS domains, BTs decompose behaviors into a hierarchical tree structure with explicit control flow. This makes action dynamics transparent and adaptable, particularly in reactive or partially observable environments. While Hierarchical Task Network (HTN) Planning also supports structured decomposition of tasks, BTs offer different execution semantics and a modular, reactive design that is often more intuitive and accessible, especially for applications requiring runtime adaptability and human interpretability.

This chapter proposes a novel algorithm for learning BTs from observed system behavior, motivated by the need for interpretable and modular representations of action models.

3.1 Related Work

Building on the theoretical foundations introduced in Section 2.3, BTs have gained prominence in planning contexts due to their modularity and hierarchical structure, making them well-suited for reactive decision-making tasks [29]. Traditionally, BTs are designed manually [108], requiring expert knowledge to construct effective behavior models. However, recent research has explored the

integration of learning techniques with BTs to automate their generation and refinement. Colledanchise et al. [30] were among the first to explore automated BT construction, introducing a genetic programming approach that initiated much of the subsequent research on learning BTs. Building on this foundation, other works investigated Reinforcement Learning (RL) (Mayr et al. [109]), Genetic Programming (GP) variants (Iovino et al. [81]), and Evolutionary Learning (Scheper et al. [150]) to generate and refine BT structures. Additional directions include Case-Based Reasoning (CBR) by Palma et al. [125] and Learning from Demonstration (LfD) by French et al. [47], both of which construct BTs from past experiences or expert demonstrations.

Most recently, Lykov et al. [105] and Styrud et al. [162] investigated the use of Large Language Models (LLMs) for BT synthesis. These works can be categorized into two main strategies: End-to-End Generation, in which fine-tuned LLMs create complete BTs from textual input, and BT Expansion, where LLMs dynamically configure or extend BTs. While LLM-based methods have demonstrated promise in generating and refining Behavior Trees from scratch, they often lack grounding in actual system behavior and remain limited in their ability to learn structured policies from real-world execution data. In contrast, this thesis focuses on learning BTs from execution traces collected during prior system runs. The objective is to infer structured, interpretable behavior models that reflect how systems operate in practice. Therefore, the discussion centers on approaches based on Learning from Demonstration (LfD) and Decision Tree (DT)-based synthesis. These methods enable structured learning from observed behavior and align well with the hierarchical and interpretable nature of BTs. LfD techniques support the derivation of behavior models from demonstrations or logs, while DT-based methods provide a data-driven way to capture conditional logic and decision structure. Together, these approaches serve as a foundation for the symbolic and structured behavior learning pursued in this thesis.

3.1.1 LfD-Based BT Learning

Learning from Demonstration (LfD) has been extensively studied by Argall et al. [6] as an efficient method for knowledge transfer. In this paradigm, *teachers* provide abundant and correct demonstrations, and *learners* correctly perceive and reproduce them. In the context of this thesis, LfD is relevant as it enables the extraction of behavior directly from execution data, an essential objective of the thesis. While applying LfD to Behavior Tree (BT) learning presents challenges, particularly in structuring demonstrations into modular and reusable components, its ability to ground behavior models in real-world traces makes it a suitable foundation for deriving interpretable and structured BTs.

Some studies have attempted to apply LfD to BT generation. Sagredo-Olivenza et al. [146] and French et al. [47] proposed approaches in which a Decision Tree (DT) is first learned as a mapping from state space to action

space and subsequently converted into a BT. While this leverages the fact that BTs generalize DTs (Colledanchise et al. [28]), the resulting BTs often become unnecessarily large and redundant. Further details on DT-based BT learning and optimization can be found in Section 3.1.2.

Robertson et al. [141] instead encoded demonstrated sequences directly into BTs without an intermediate DT step, structuring each demonstration as a sequence node and grouping multiple sequences under a fallback node. This guaranteed completeness but produced extremely large trees (often exceeding 50,000 nodes) that were difficult to interpret. Furthermore, this approach lacks dynamic adaptability, as all actions and conditions are executed in a predefined order, limiting the ability of the learned BTs to react efficiently to environmental changes. A further direction was proposed by Suddrey et al. [163], who generated BTs directly from natural language task descriptions using a database of predefined structures. If no relevant BT is found, a new tree is assembled by matching parsed expressions to predefined primitive structures. While this approach allows for a broad range of task definitions, it remains unclear how it would handle tasks requiring precise spatial reasoning, such as assembly operations where object positions and relations are critical.

This thesis builds on these efforts by introducing BT-Factor, a novel approach that learns BTs from previously executed plans, using execution traces rather than relying solely on demonstrations or predefined mappings. Learning from recorded execution traces of past plans can be viewed as a form of learning from observations, allowing the system to generalize behaviors and adapt to similar tasks.

3.1.2 Learning Compact BTs Using DTs

DT-to-BT conversion approach has been applied in different domains. In the video game context, Sagredo-Olivenza et al. [146] used it to assist game designers in defining behaviors for Non-Player Characters (NPCs). A DT was trained to map the game’s state to NPC actions, then simplified and converted into a BT. In robotics, French et al. [47] extended this approach to a mobile manipulator performing a house-cleaning task.

Algorithms such as C5.0 (an extension of C4.5) [135] and CART [17] are widely used for DT construction. These differ in how they split attributes, prune trees, and structure the final model. While C5.0 uses Information Gain (Entropy) for splitting, CART relies on the Gini index. Their pruning strategies also differ, with C5.0 applying Binomial Confidence Limit pruning post-construction and CART employing Cost-Complexity pruning during tree formation. Additionally, C5.0 allows both multi-way and binary splits, whereas CART strictly produces binary trees [126].

While DT-to-BT conversion has enabled structured policy extraction, existing methods often fail to leverage the hierarchical modularity of BTs. One ma-

major limitation is that predicates are redundantly evaluated at multiple depths within the tree, leading to larger and less interpretable BTs [29].

To address this issue, several optimization techniques have been proposed. Sagredo-Olivenza et al. [146] introduced manual redundancy reduction, whereas French et al. [47] applied the Espresso logic minimization algorithm [16] to streamline BT structures. Building on this, Wathieu et al. [171] developed RE:BT-Espresso, an algorithm that removes redundant logic and identifies common factors to improve BT compactness and interpretability. However, these methods remain limited by the simplicity of their logical reduction techniques and fail to fully optimize BT efficiency. More recently, Scheide et al. [149] introduced a method that uses Boolean algebra simplification, specifically the Quine-McCluskey algorithm [134], to generate compact BTs.

This thesis addresses these limitations by introducing BT-Factor, a novel approach that applies logic factorization techniques inspired by circuit design to the problem of BT optimization. BT-Factor exploits advances in Boolean expression minimization to produce BTs that are both compact and interpretable, without sacrificing efficiency.

The minimization of Boolean expressions has been extensively studied in circuit design, where finding an optimal factored form is NP-hard [114]. To address this complexity, heuristic algorithms have been developed to obtain efficient representations. Existing strategies include graph partitioning [56], algebraic division, where Boolean expressions are treated as polynomials over real numbers, allowing for simplifications through “weak divisions” [170], and Boolean division, which fully exploits Boolean properties but often incurs higher computational costs [159]. Building on these insights, BT-Factor adapts factorization methods from circuit optimization to the synthesis of BTs, reducing redundancies and improving structural clarity. In doing so, it advances the interpretability of learned BTs while maintaining their efficiency and compactness.

3.2 BT-Factor

The previous sections have highlighted key challenges in learning Behavior Trees from demonstrations, particularly issues related to redundancy, scalability, and interpretability in DT-based BT conversion. To address these limitations, this thesis introduces BT-Factor.

BT-Factor, illustrated in Figure 3.1, draws inspiration from LfD. Rather than restricting LfD to natural human demonstrations, the approach extends the possible input to traces from any executed plans, enabling the utilization of historical data. The objective is to extract and formalize relationships among variables and their connections to logical elements within the application domain through a BT, which serves as a human-interpretable structure.

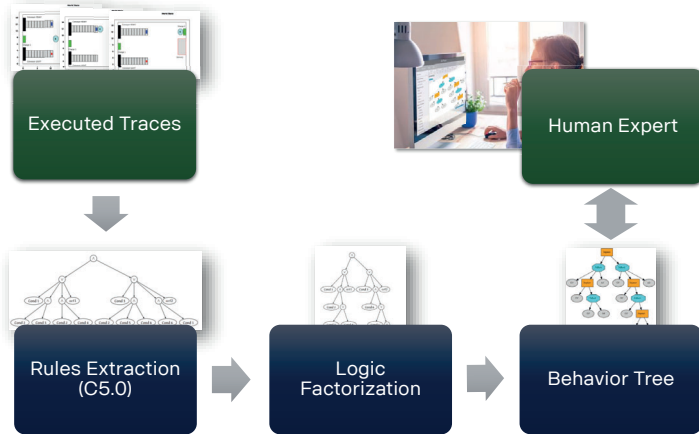


Figure 3.1: BT-Factor inputs historical executed traces to learn the implicit knowledge of planning experts by using C5.0 and logic factorization methods. The output is a Behavior Tree, which can be inspected by human experts.

This facilitates human involvement at the final stage of BT generation by providing a readable and organized output, allowing users to comprehend and refine the results of the learning process. The approach is based on the premise that incorporating human knowledge and experience into learning techniques enhances the overall outcome [173].

To achieve this, the C5.0 decision tree algorithm [95] is used to infer the knowledge implicitly encoded in previous plans. The underlying logic is represented using Horn clauses and refined through logic factorization algorithms [170], ensuring a compact and interpretable BT.

3.2.1 Overview

BT-Factor is a modular method consisting of four core stages: **Rule Extraction**, **Rule Conversion**, **Rule Factorization**, and **BT Conversion**. These modules operate sequentially to extract logical rules from data, refine them into a structured format, apply factorization to eliminate redundancies, and ultimately construct a compact BT. Table 3.1 summarizes the main differences between BT-Factor and RE:BT-Espresso [171], the closest state-of-the-art method. Unlike RE:BT-Espresso, which employs CART for rule extraction, BT-Factor uses C5.0. The comparison between C5.0 and CART [126] shows that the former is more accurate. Indeed, provides better handling of multi-way splits and more effective pruning strategies, leading to more compact and structured rules. Additionally, while RE:BT-Espresso represents rules only in

DNF, BT-Factor allows both CNF (Horn clauses) and DNF, making it more adaptable to different logic structures. The key improvement lies in the rule factorization approach: while RE:BT-Espresso applies a basic common factoring technique, BT-Factor leverages GFACTOR, a more advanced logic factorization approach, which improves the compactness and efficiency of the generated BT.

Table 3.1: Overview of BT-Factor core modules

	Description	BT-Factor	RE:BT-Espresso[171]
Rule extr.	Learn logic rules from data	C5.0	CART (scikit-learn)
Rule conv.	Express the rules into a canonical form.	CNF (Horn clauses) or DNF	DNF
Rule fact.	Compact boolean expressions by removing redundancies	GFACTOR	Common factoring
BT conv.	Replicate the rule logic into a BT	naive DT-to-BT	naive DT-to-BT

Running Example: Battery Management Scenario

The battery management scenario, used to illustrate BT-Factor, is a simplified version of a more complex logistics simulation used in the experiments (Section 3.3.1). It features an autonomous agent (an electric truck) that operates between a charging station (C) and two loading stations ($L1$, $L2$). The primary objective in this scenario is to learn a BT that governs the agent's battery-related behavior. To this end, only two actions are considered:

- *go to charge*: navigate the agent to the charging station C ,
- *charge*: charge the battery agent to 100%

These are the only actions for which decision rules are to be extracted.

The literals (i.e., variables) used in this scenario are:

- *battery*: a continuous variable representing the current battery level of the truck, expressed as a percentage (0-100%).
- *at*: a categorical variable indicating the truck's current location. Possible values are the charging station (C) and the two loading stations ($L1$, $L2$).

- *weight*: a continuous variable representing the current load of the truck, expressed as a percentage of its maximum transport capacity.

This abstraction allows for a clear illustration of BT-Factor’s learning stages, without the complexity of full environment variability. Specifically, the loading stations are treated identically, and the agent’s movement between locations is abstracted as a single-step action.

The scenario, schematically illustrated in Figure 3.2, serves to demonstrate each core module of BT-Factor step by step, showing how logical rules are extracted, refined, factorized, and translated into a compact BT.

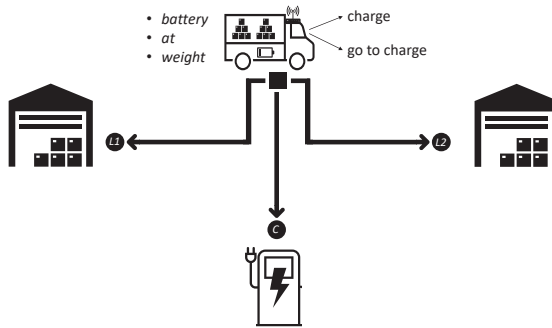


Figure 3.2: Simplified environment with one truck moving between a charging station (C) and two loading stations (L1, L2).

3.2.2 Rule Extraction

The first step in BT-Factor is Rule Extraction, where executed traces serve as input to the C5.0 decision tree algorithm, with the executed action used as the target attribute. C5.0 generates a decision tree classifier, which can be transformed into a collection of if-then rules. These rules represent the decision-making logic extracted from historical executions. To enhance compactness and efficiency, the extracted rules are then expressed in logical form and minimized using the Espresso algorithm [16], which eliminates redundancies and simplifies Boolean expressions. Each extracted rule consists of conditions that correspond to literals j (i.e., atomic logical statements). To quantify the importance of each literal, we define its frequency f_j as the number of occurrences of that literal in the rule set. For each rule r , the set of literals L_r is identified, and an average score s_r is computed based on the literal frequencies:

$$s_r = \frac{\sum_{j \in L_r} f_j}{|L_r|}.$$

Rules are then sorted in descending order based on s_r , ensuring that higher-priority rules appear earlier. Since a Behavior Tree (BT) executes its children in depth-first order, placing the most important rules on the leftmost branches optimizes execution efficiency by evaluating critical conditions first.

Example: The extracted rules in the battery management scenario are:

- Rule 1: $(at\ L1 \wedge battery \leq 10) \vee (at\ L2 \wedge battery \leq 10) \vee (at\ L1 \wedge weight > 50 \wedge battery \leq 20) \vee (at\ L2 \wedge weight > 50 \wedge battery \leq 20) \rightarrow go\ to\ charge$
- Rule 2: $at\ C \wedge battery \leq 99 \rightarrow charge$

The average scores are computed as:

- $s_1 = \frac{10}{5} = 2$
- $s_2 = \frac{2}{2} = 1$

Since Rule 1 has a higher score, it is positioned on the left side of the resulting BT. This ordering ensures that the decision to move toward charging is prioritized over the decision to charge at the station.

3.2.3 Rule Conversion

Once logical rules are extracted, they are converted into a conjunction of Horn clauses (HCs). This transformation preserves the logical structure of the rules while organizing them into a tree representation where the root is a conjunction node, and each subtree, rooted in a disjunction, corresponds to a separate rule. This, in combination with rule ordering, is one key factor in making the final BT more efficient in execution. To further enhance compactness, the default action rule is identified and removed from the rule set. Instead, its corresponding action is added as the last child of the root, ensuring that the BT remains compact. Note that this step is applicable in the case of mutually exclusive rules and, therefore, reasonably connected by a disjunction. In the case of concurrent rules, these can be described in the form of conjunctions and executed in parallel as proposed by French [47].

Example Applying Horn clause conversion to the battery management scenario yields the following Horn Clauses (HCs):

- HC1: $(\neg at\ L1 \wedge \neg at\ L2) \vee (battery > 10 \wedge battery > 20) \vee (weight \leq 50 \wedge battery > 10) \vee go\ to\ charge$
- HC2: $\neg at\ C \vee battery > 99 \vee charge$

3.2.4 Rule Factorization

Extracted rules often contain repeated conditional statements, leading to redundancy and inefficiency. To address this, factorization techniques are applied to simplify logical expressions while maintaining their correctness.

The RE:BT-Espresso algorithm [171] performs factorization via the distributive law, extracting common conditions and placing them under a Sequence node in the BT. However, this approach is limited when repeated conditions do not appear in every conjunction, resulting in suboptimal factorization. To overcome this limitation, GFACTOR [170] is used. This method applies a recursive factorization approach where a Boolean expression f is decomposed into $p \cdot q + r$ where p , q , and r are Boolean expressions treated as polynomials. The decomposition is recursively applied to p , q , and r , selecting as heuristic the condition that occurs in the greatest number of conjunctions as the best one to factor out.

Algorithm 2 describes the GFACTOR method, which systematically reduces rule complexity using the following key functions:

- `MOST_COMMON_CONDITION(f)` returns the condition that occurs in the greatest number of conjunctions of f . If there is not a dominant condition (the number of occurrences of the most common condition is 1), it returns 0.
- `DIVIDE(f, d)` performs the algebraic division between f and d and returns the quotient q and the remainder r .
- `MAKE_CUBE_FREE(f)` removes the common cubes from f making it cube free.
- `CUBE_FREE(f)` returns *True* if the expression f is cube free, *False* otherwise.
- `COMMON_CUBE(f)` returns the largest common cube of f .

Example Applying GFACTOR to the battery management case leads to:

- HC1: $C_3(C_4 + C_5) + (C_1 \cdot C_2) + A_0$
- HC2: $C_6 + C_7 + A_1$

where $C_i = \neg c_i \forall i$ and the corresponding literals are listed in Table 3.2.

Note that when applying RE:BT-Espresso, HC1 is not factorized as there are no common repeated conditions.

Algorithm 2 Factorizing a Boolean expression f by heuristically minimizing the number of literals

Require: f ▷ Boolean expression to be factorized

Ensure: Factorized f

```

1: function GFACTOR( $f$ )
2:    $d \leftarrow$  MOST_COMMON_CONDITION( $f$ )
3:   if  $d = 0$  then
4:     return  $f$ 
5:    $q, r \leftarrow$  DIVIDE( $f, d$ )
6:   if  $|q| = 1$  then
7:     return LF( $f, q$ )
8:    $q \leftarrow$  MAKE_CUBE_FREE( $q$ )
9:    $d, r \leftarrow$  DIVIDE( $f, q$ )
10:  if  $d = 0$  then
11:    return  $f$ 
12:  if CUBE_FREE( $d$ ) then
13:     $q \leftarrow$  GFACTOR( $q$ )
14:     $d \leftarrow$  GFACTOR( $d$ )
15:    if  $r \neq \emptyset$  then
16:       $r \leftarrow$  GFACTOR( $r$ )
17:    return  $q \cdot d + r$ 
18:   $c \leftarrow$  COMMON_CUBE( $d$ )
19:  return LF( $f, c$ )

```

```

20: function LF( $f, l$ )
21:   $q, r \leftarrow$  DIVIDE( $f, l$ )
22:   $q \leftarrow$  GFACTOR( $q$ )
23:  if  $r \neq \emptyset$  then
24:     $r \leftarrow$  GFACTOR( $r$ )
25:  return  $l \cdot q + r$ 

```

Table 3.2: Logical conditions and symbols in battery management example

Symbol	Condition
c_1	at $L1$
c_2	at $L2$
c_3	battery ≤ 10
c_4	battery ≤ 20
c_5	weight > 50
c_6	at C
c_7	battery ≤ 99
A_0	go to charge
A_1	charge

3.2.5 BT Conversion

The factorized logic rules are converted into a BT by using the naive Decision Tree to Behavior Tree algorithm proposed by French et al. [47], whereby conjunctions are encoded as Sequence nodes and disjunctions as Fallback nodes. When the rules are mutually exclusive, the resulting BT has a Sequence root node. In this case, the set of expressions forms a tautology, ensuring that only one path through the logic tree is true at any given time. On the other hand, if multiple actions can be performed simultaneously, the rules are described in the form of conjunctions, leading to a BT with a Parallel root node. Applying BT-Factor significantly reduces the number of nodes in the resulting BT by eliminating redundant conditions.

Example In the battery management example, BT-Factor produces a tree with seven fewer nodes than RE:BT-Espresso. Figure 3.3 shows the subtrees for Rule 1 (*go to charge*) learned with both methods.

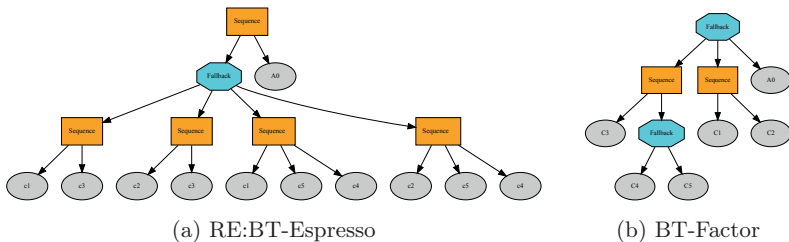


Figure 3.3: Learned subtree for the *go to charge* action with RE:BT-Espresso (left) and BT-Factor (right). Both encode the same behavior, but BT-Factor factorizes repeated conditions, resulting in a more compact and readable tree. Note that the total number of nodes has decreased from 17 to 10.

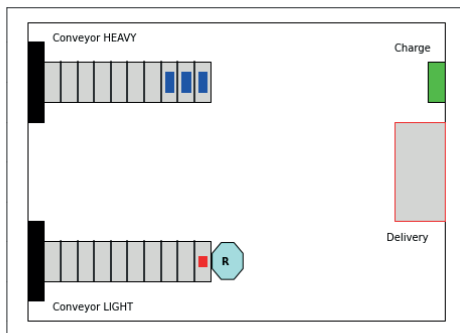


Figure 3.4: Simulated environment for Setup I, representing a scenario with an autonomous agent equipped with a battery, two loading areas (conveyor heavy and conveyor light), a depot (delivery), and a charging station (charge)

3.3 Experimental Set Up and Results

We have implemented BT-Factor in Python3.8, with the decision tree component developed in R. The full source code is available online¹ to support reproducibility. We have conducted an empirical evaluation has been conducted across multiple experimental setups.

The analysis compares BT-Factor with RE:BT-Espresso [171]. It is important to note that RE:BT-Espresso does not determine the appropriate pruning level of the extracted BT; instead, it generates multiple BTs and requires the user to manually select the most suitable one. For each experiment, the criteria for selecting the BT used in the comparison are explicitly defined.

3.3.1 Setup I: Industrially-Relevant Simulation Environment

Setup I extends the simplified battery management scenario presented earlier (Section 3.2.1) by incorporating additional environment complexity and a broader set of behaviors. While the basic structure of an agent navigating between locations and managing its battery is preserved, this setup introduces new decision points and more realistic operating conditions. In particular, the environment, shown in Figure 3.4, simulates a logistics scenario inspired by industrial mining applications². The agent operates across five locations: two loading areas (handling heavy and light material), a depot (delivery), and a

¹At <https://github.com/SimonaGug/BT-from-planning-experts>

²Adapted from <https://github.com/jstyrud/WASP-CBSS-BT>

charging station. Each loading area is specialized for a specific material type, which is randomly generated at each time step with probabilities of 12% and 6%. This stochastic process ensures that each simulation instance presents different conditions for the autonomous agent. Compared to the simplified version, Setup I includes:

- Additional actions, such as: *move to conveyor light*, *move to conveyor heavy*, *move to delivery*, *pick*, *place*, and *idle* in addition to the original battery related actions (*move to charge* and *charge*).
- New literals, including: *conveyor light* and *conveyor heavy*, representing the number of boxes on the respective conveyors, as well as the extended version of *at*, which now includes the delivery station location, alongside the previously used *battery* and *weight*.

The agent’s battery decreases by one unit per time step, with each action (except *idle*) incurring an additional unit cost. A manually designed BT is used as input to the simulator, generating 100 execution traces, each consisting of 200 ticks. At each time step t , the BT executes a single tick, and all the environment variables are updated accordingly. To capture high-level execution snapshots, all actions and environment state variables are logged in a `.csv` file for each tick.

The dataset is processed using the C5.0 rule-based algorithm in \mathbb{R}^3 , employing default hyperparameters. Extracted rules are stored in a `.txt` file, which is used by the pipeline in Python to extract the BT. This setup primarily evaluates the correctness of the learned behavior, the extracted rules’ accuracy, and the performance of the inferred BT.

The evaluation is based on a fitness function, similar to the one defined by Iovino et al. [81], that measures the efficiency of the BTs in terms of material delivered. In detail, each delivery of light and heavy material obtains a score of 1 and 2, respectively. On the other hand, for each time step in which a loading area is full when the material would be otherwise generated, there is a negative term of -0.5 for the blocked generation of heavy material and -0.25 for light material.

Correctness is evaluated based on whether the agent successfully executes a policy during the entire duration of the simulation (200 ticks), regardless of the performance obtained. In detail, if the agent has zero battery and is not located at the charging station, the learned rules describing the implicit logic of the domain action are considered incorrect. The results of 200 simulations are summarized in Table 3.3.

For each simulation, the fitness values of all trees generated by RE:BT-Espresso are computed, and the tree with the highest fitness value is selected

³<https://CRAN.R-project.org/package=C50>

for comparison. When multiple trees, corresponding to different pruning levels, have the same highest fitness score, the one with the highest pruning level is selected. However, the level of pruning can affect the correctness of the learned behavior. To account for this, correctness is also evaluated for RE:BT-Espresso BTs without pruning, as shown in Table 3.3.

BT-Factor achieves 100% correct behavior, whereas RE:BT-Espresso exhibits correct behavior in only 62.5% (125 simulations) of cases when selecting the highest-fitness BT and 72% (144 simulations) when no pruning is applied. Note that a high fitness score does not imply correctness.

Table 3.3: Correctness experiments. Higher values are better.

Method	% correct behavior
BT-Factor	100%
RE:BT-Espresso	62.5%
RE:BT-Espresso (no pruning)	72.0%

Performance is evaluated by comparing fitness scores obtained using BT-Factor and RE:BT-Espresso in the same 200 simulations used to evaluate correctness. For each simulation, when running RE:BT-Espresso, the BT with the best fitness score among all of the generated BTs (at all pruning levels) is considered. Recall that not all of the BTs obtained by RE:BT-Espresso lead to correct behavior. Limiting the evaluation to cases where correct behavior is achieved reveals that BT-Factor outperforms RE:BT-Espresso. As shown in Table 3.4, BT-Factor achieves higher fitness scores than RE:BT-Espresso. Moreover, the BTs learned using BT-Factor lead the agent to achieve the same performance as in the original BT from which the training data was obtained. Furthermore, when considering only the 125 simulations where RE:BT-Espresso results in correct behavior, the average fitness score is higher and the performance more stable (lower variance) than when all simulations are considered, indicating greater stability.

Table 3.4: Performance experiments when using BT-Factor and RE:BT-Espresso on 200 simulations and when considering only the 125 where RE:BT-Espresso learned correct behaviors (in parentheses). Higher mean values entail better performance, lower variance values entail more stability.

	Mean	Variance
Original BT	38.505 (37.928)	22.642 (24.151)
BT-Factor	38.505 (37.928)	22.642 (24.151)
RE:BT-Espresso	32.433 (36.704)	108.348 (24.084)

Graph Edit Distance. RE:BT-Espresso uses the Graph Edit Distance (GED) score to evaluate the quality of the BT by assessing the structural similarity between learned and reference BTs. In this setup, the GED of the obtained BTs is computed following the approach outlined by Wathieu et al. [171]. Each subtree of the learned BT is compared against each of the four subtrees of the original manually designed BT from which the training data was obtained. The results are presented in Table 3.5. Notably, the GED score of the BT learned using Horn clauses is even lower than the BT learned without Horn clauses, despite the two trees describing the exact same behavior, as the actions are mutually exclusive. In fact, the GED measures the similarity between two graphs, regardless of the semantics. However, in this context, the semantics cannot be neglected as the goal is that the learned tree faithfully reproduces the behavior, regardless of the similarities with the original graph. Therefore, GED results are computed and presented for completeness, but caution is advised against relying solely on GED as a measure of BT quality, as it does not fully capture correctness, performance, or behavioral equivalence.

Table 3.5: Graph Edit Distance (GED) between original and learned BT with different methods for each subtree (sub) of the original BT. Lower values are better.

	sub1	sub2	sub3	sub4
BT-Factor (Horn clauses)	12.0	7.0	10.0	11.0
BT-Factor	14.0	7.0	14.0	12.0
RE:BT-Espresso	16.0	9.0	15.0	13.0

3.3.2 Setup II: Synthetic Data Simulator

The factorization algorithm (Algorithm 2) is integrated into a modified version of the RE:BT-Espresso simulator⁴, which builds upon the original RE:BT-Espresso implementation⁵, including 100 synthetic BTs provided by Wathieu et al. [171]. Given an input BT, the simulator generates random values with the range [0.0, 1.0] for each measurable condition, logging all actions and environmental states per tick into a `.csv` file, thus generating synthetic plan execution traces. This setup is used to evaluate the effects of the logic factorization on different BT structures.

A detailed analysis of the **tree structure** is conducted using this setup. Since previous results indicate that unpruned BTs better preserve behavior,

⁴Modified RE:BT-Espresso version with GFACTOR integration and experimental scripts: <https://github.com/SimonaGug/RE-BT-Espresso>

⁵Original RE:BT-Espresso repository by Wathieu et al. [171]: <https://github.com/interaction-lab/RE-BT-Espresso>

comparisons with RE:BT-Espresso are performed using its unpruned BTs. Notably, this setup does not provide a direct means to verify correctness, as it relies on a synthetic benchmark not representing any real-world behavior.

The 100 synthetic BTs⁵ are used to learn BTs, which are then post-processed by applying the GFACTOR (Algorithm 2). In one case, the resulting BT had only one node and was therefore excluded from the analysis. For the remaining 99 synthetic BTs, 11 simulations are run per BT. The evaluation is based on the following parameters:

Number of total nodes. As stated by Wathieu et al. [171], a reduction in BT size can contribute to improved interpretability. For this reason, the percentage of nodes removed when post-processing learned BTs with the GFACTOR algorithm is measured. Figure 3.5a shows the average percentage of nodes removed, along with standard deviations, for each learned BT across 11 simulations. The results indicate that, among all the learned BTs, the size has been reduced with a mean of 11.63%, thus making the BTs more compact as a result of using GFACTOR.

Number of Condition nodes vs. Fallback nodes. Given a fixed number of condition nodes, an increase in Fallback nodes generally leads to fewer (or equal in the worst scenario) conditions being ticked during execution, improving efficiency. Therefore, the condition-to-fallback ratio is considered to assess this, and it is defined as follows:

$$cf = \frac{\text{num_conditions}}{\text{num_fallbacks}}$$

Lower values of cf indicate higher efficiency, either due to fewer conditions being checked or the presence of more alternative courses of action. This coefficient is used to compare the structure of the learned tree. In detail, we compute the cf ratio for the tree learned with the original RE:BT-Espresso and the one learned with RE:BT-Espresso enhanced with the proposed factorization algorithm. The average percentage reduction in cf when post-processing the learned BTs with the GFACTOR algorithm is shown in Figure 3.5b. Among all the learned BTs, the ratio is reduced with a mean of 69.34%. This result, together with the previous one, shows that the factorization algorithm leads to a reduction of condition nodes and/or an increase of fallback ones, thus making the BT execution more efficient.

3.3.3 Setup III: Pick-and-Place Demonstrations

This scenario consists of a workspace where three video demonstrations of a human performing a pick-and-place task with a cube. The hand pose and the cube pose are extracted, using MediaPipe and Aruco markers, respectively, as

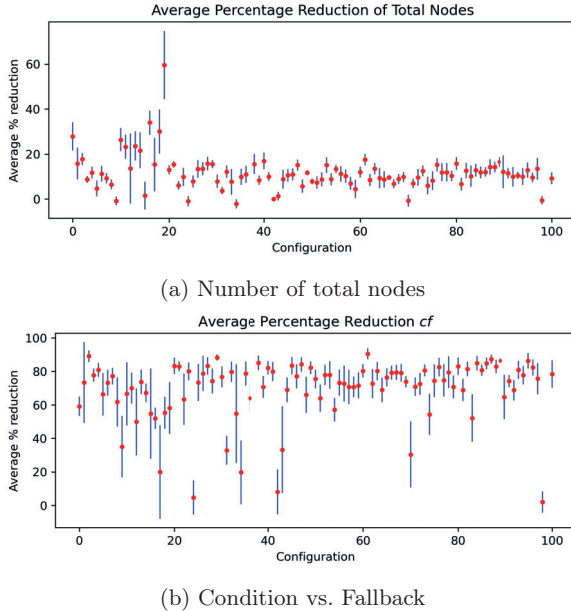


Figure 3.5: Average percentage of node (above) and cf (below) reduction (red) together with their standard deviations (blue) for each learned BT when applying GFACTOR. The average percentage of node reduction is computed as $\frac{\sum_{i=0}^s node_reduction_i}{s}$, where j is the number of one of the 99 synthetic BTs, s is the total number of simulations (in our experiments 11) and $node_reduction_i = \frac{tot_nodes(RE:BT)_i - tot_nodes(RE:BT+GFACTOR)_i}{tot_nodes(RE:BT)_i} \cdot 100$. The same reasoning applies to the computation of the cf reduction.

state variables. Each time step t , is manually labeled with the action performed (pick, move, place). The objective is to learn a BT that can be executed by a robotic manipulator to perform the Pick-and-Place task. This setup aligns with Learning from Demonstration (LfD) applications [171].

This setup is used to compare the ability of RE:BT-Espresso and BT-Factor to generalize the noisy three demonstrations. The main challenge when using RE:BT-Espresso is to manually select the best BT among all the generated pruning levels. If pruned too much, the tree oversimplifies and does not reflect the correct behavior. More conservative pruning leads to significantly more nodes. To select the best one, we compute the percentage of misclassified cases of all RE:BT-Espresso trees. We select the BT with the smallest error (3.3%). However, the BT-Factor generated tree achieves a better result (1%). In this empirical example, BT-factor’s performance is better than the human-selected

Espresso tree with a similar number of nodes. In this example, the advantage of BT-Factor did not come from the factorization as the trees are fairly small, but from the different decision tree learning algorithm.

Noise Robustness

For BT-Factor to be applicable in real-world scenarios, it must demonstrate robustness to noisy data, as real-world execution traces are inherently subject to sensor inaccuracies, environmental fluctuations, and data inconsistencies.

To simulate real-world imperfections, noise was introduced by perturbing state variables with random deviations, ensuring a realistic representation of sensor errors and environmental uncertainties. The noise followed a controlled distribution, progressively increasing to assess the breaking point where the model's accuracy begins to decline.

Figure 3.6 illustrates the effect of increasing noise levels on accuracy and the number of extracted rules. The results show that C5.0 maintains high accuracy (above 95%) when the signal-to-noise ratio (SNR) exceeds 5, indicating that the method effectively generalizes under moderate noise conditions. As noise increases beyond this threshold, accuracy drops sharply, while the number of extracted rules rises significantly. This suggests that C5.0 discards outliers effectively when $\text{SNR} > 5$ but begins to overfit when noise becomes dominant, learning spurious patterns instead of meaningful action rules.

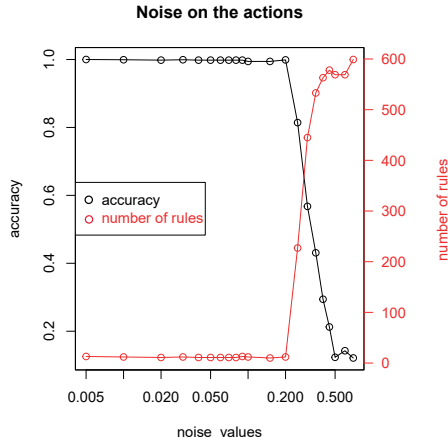


Figure 3.6: Impact of noise on accuracy and the number of extracted rules. Accuracy stays above 95% for $\text{SNR} > 5$, but drops sharply with higher noise, while rule count increases, indicating overfitting. BT-Factor filters out moderate noise but struggles to generalize when noise levels become too high.

These findings highlight the importance of noise tolerance for deploying BT-Factor in dynamic, real-world environments, where perfect data is rarely available.

3.4 Summary and Outlook

This chapter introduced BT-Factor, a novel method for learning BTs from executed logs. The approach integrates decision tree learning, logic-based rule conversion, and advanced factorization techniques to produce interpretable and compact BTs. Through a step-by-step example and evaluation across both simplified and industrially relevant setups, the method was shown to reduce redundancy and improve interpretability while preserving decision fidelity.

Importantly, the results also demonstrated BT-Factor’s capacity to handle noise and variability in execution data, an essential property for real-world deployment.

While this chapter focused on how to learn Behavior Trees from execution traces, it has also emphasized the value of BTs as a human-interpretable and modular representation of system behavior. However, this raises an important question: how can the quality of learned BTs be evaluated, particularly with respect to their intended advantages, such as interpretability, modularity, and correctness?

The next chapter addresses this challenge by critically examining how BTs can be systematically evaluated. It explores existing gaps in the literature and proposes both new and adapted metrics to support more rigorous and meaningful assessment of learned BTs.

Chapter 4

Evaluating Behavior Trees

The previous chapters introduced Behavior Trees (BTs) as a structured, modular, and interpretable representation of system behavior, and presented BT-Factor as a method for learning BTs from execution traces. This learning process is motivated not only by the ability to automate policy acquisition but also by the claim that BTs, by virtue of their structure and semantics, are more understandable, adaptable, and suitable for human interpretability and evaluation than alternative representations. Yet, despite the potential of BTs, assessing the quality of a learned BT in a principled and meaningful way remains a critical and underexplored challenge. The value of a learned BT is not limited to whether it executes correctly. Its utility also depends on properties such as interpretability, reactivity, modularity, and robustness, properties that are often invoked in literature as key strengths of BTs. Despite the widespread adoption of BTs in robotics and related fields, there exists no systematic and widely accepted framework for evaluating these properties. In many cases, evaluation relies on ad hoc metrics, qualitative judgments, or assumptions that are not empirically validated. For example, node count is frequently used as a proxy for interpretability, despite the lack of evidence linking structural size to human understanding. Similarly, structural similarity metrics such as Graph Edit Distance (GED) are often used without capturing the functional behavior or semantic relevance of BT components.

This chapter addresses this gap by conducting a systematic investigation into how learned BTs can be evaluated in a way that reflects both their functional performance and their qualitative strengths. The goal is to develop a framework for BT evaluation that is grounded in rigorous definitions, supported by appropriate quantitative metrics, and aligned with the practical concerns of using BTs as decision-making policies in real-world systems.

To that end, the chapter begins by identifying and critically reviewing existing evaluation approaches, including both structural and behavioral metrics. It then introduces a set of well-defined BT properties drawn from the literature,

followed by corresponding quantitative measures designed to evaluate these properties in practice. A particular focus is given to interpretability, which is further investigated through an empirical user study involving human participants. The findings contribute to a deeper understanding of what makes a BT “good” from the perspectives of both machine and human interpretability.

Through this work, the chapter aims to provide a foundation for standardized and reproducible evaluation of BTs. This not only supports the validation of learning methods such as BT-Factor but also contributes to the creation of behavior models that are transparent, modular, and verifiable, qualities that are essential for real-world deployment and effective human interaction.

4.1 Related Work

Existing surveys on BTs in robotics [80, 122] and introductory overview texts [29] primarily focus on formalizing, classifying and categorizing existing works, and on providing recommendations of best practices when applying BTs to specific problems. While these works serve an important function in the community, they do not sufficiently address the questions of measuring, evaluating, and comparing BT structural and behavioral properties of BTs.

Beyond general overviews, a smaller body of work has attempted to compare BTs to alternative control architectures, most notably Finite-State Machines (FSMs). Iovino et al. [79] directly compare BTs and FSMs in the context of mobile manipulation, proposing metrics for properties such as reactivity, modularity, readability, and design complexity. These metrics are motivated by the need for fair comparison and are designed to be applicable across architectures. In a related contribution, Iovino et al. [78] focused specifically on quantifying development effort, arguing that the higher modularity of BTs significantly reduces the time and operations needed to modify and adapt behaviors. They introduced a graphical metric based on counting the modifications necessary to reflect new requirements, which illustrates how structural flexibility can provide practical design advantages. In a more formal line of work, Biggar et al. [13] present a theoretical framework for comparing the expressive power of BTs with other action-selection mechanisms, including Decision Trees (DTs), Teleo-reactive Programs (TRs), and FSMs. These comparisons underscore the importance of principled evaluation when selecting or designing policy representations.

While such comparative works provide valuable insights, they primarily focus on cross-architecture evaluation and often adopt metrics tailored for that purpose. What remains largely missing is a systematic framework for evaluating BTs in their own right, based on their internal properties and their intended advantages.

In the following sections, several individual studies are presented that have attempted to fill this gap by proposing metrics for specific BT properties. How-

ever, these works typically focus on a single property at a time, introducing custom metrics that are rarely validated across contexts or reused by other researchers. The fragmented nature of these efforts limits their generalizability and makes it difficult to form a cohesive understanding of BT evaluation. This lack of standardization is a limiting factor within the community, as different sub-communities, or sometimes even different research groups, might have distinct and even irreconcilable definitions for commonly used terms referring to BT properties. Addressing this gap requires a shift in focus, from surveying applications and structural variants of BTs to systematically analyzing how their properties are defined and assessed.

4.2 Metrics

This section provides a comprehensive overview of the metrics employed to assess BTs, drawing from both existing literature and newly formulated metrics. Metrics are categorized into *Functional* and *Non-Functional*, each serving distinct evaluation purposes.

4.2.1 Functional Metrics

Functional metrics assess whether a given BT satisfies the desired requirements and executes its intended functions.

Use of Resources

This metric evaluates the optimization of resource allocation, management, and utilization by a robot in the execution of its designated task. One commonly considered resource is execution time, which can refer to the overall duration required for a BT to complete its execution [142, 24] or the duration of a single node execution. This is particularly useful in assessing the impact of introducing new specific nodes [25].

Predictability Distance

Originally defined for evaluating parallel behaviors [26, 25], this metric measures the deviation between expected and actual execution progress. A target progress value is set, and the metric computes the average deviation between the expected and the actual time at which the action reaches progress closest to the desired value.

Success Rate

Success rate can be analyzed in different ways, depending on the level of granularity desired:

- **Desired Behavior Success Rate** evaluates the extent to which the behavior executed by the BT aligns with the behavior expected by a behavior oracle. It focuses solely on the desired behavior while disregarding the internal mechanisms of the BT itself (e.g., return status). For its flexibility and effectiveness, this metric stands out as the foremost and prevalent approach when referring to success rates [127, 38].
- **Accuracy for behavior usage** originally defined by Pereira & Engel [132] as the ratio between the correct activations of the inner sub-behaviors over the total expected activations. BTs typically consist of multiple modular components, this metric is particularly useful for evaluating whether specific branches or behaviors are activated as intended.
- **Single Node Success Rate** refers to the ratio between the number of successful completions of a node $n \in N$ (i.e., when it returns *Success*) and the total number of executions of the node, including both successful and failed attempts, denoted by n_S and n_F respectively. Mathematically, it is expressed as:

$$SR_n = \frac{n_S}{n_S + n_F}$$

Unsafe State Count

This metric quantifies the frequency at which an agent enters unsafe states. According to the definition of *safety* (Sec 4.3, Def 4.3.3), a state is unsafe if it leads to irreversible undesired behaviors within a system or environment. Given a set of unsafe states U , the Unsafe State Count metric is defined as:

$$USC = \frac{n_U}{n_T}$$

where n_U is the number of unsafe state occurrences and n_T is the total number of states visited during execution. A higher USC value indicates that the agent frequently encounters hazardous states, thus leading to potential safety issues. This metric does not correlate directly with success rates but rather provides a complementary safety assessment.

Condition Check Frequency

Condition Check Frequency quantifies the frequency at which a specific Condition Node is evaluated (i.e., ticked) over a given time interval:

$$CCF = \frac{n_evaluations}{\Delta T},$$

where $n_evaluations$ is the number of times a specific condition is checked and ΔT is the considered time interval. A higher CCF indicates that a certain Condition is checked more often, resulting in more timely updates of the BT decision-making process in response to environmental changes.

4.2.2 Non-Functional Metrics

Non-functional metrics measure aspects of the BT that are independent of its functional performance but provide insights into structure, efficiency, and interpretability.

Tree Dimensions

BT dimensions provide insights into tree structure and complexity. Three key aspects are:

- **Tree Height** is the total number of edges from the root node to the leaf node in the longest path [31].
- **Tree Width** is the maximum number of nodes that exist at any level within the tree structure [124].
- **Number of Nodes** is the total count of individual elements (nodes) in a BT, often used to evaluate interpretability [171, 141]. Note that this metric does not distinguish between different types of nodes in a BT, such as control and execution nodes, and instead provides a holistic count of all nodes within the BT.

Graph Edit Distance (GED)

It measures the similarity between two graphs, by computing the minimum cost of transforming one graph into another through node and edge modifications [147]. Wathieu et al. [171] use GED to evaluate the degree of similarity between two BTs, while Iovino et al. [78] use it to evaluate the difference in modularity between a BT and a Finite State Machine (FSM) architecture.

Condition-to-Fallback ratio (CFR)

This has been introduced and used in our prior work (Gugliermo et al. [65]), which forms part of the contributions of this thesis, as an indicator of the *efficiency* of BT execution. It is defined as the ratio between the total number of condition nodes and the total number of fallback nodes in a BT:

$$CFR = \frac{num_conditions}{num_fallbacks + \epsilon},$$

where ϵ is an infinitesimal value to prevent division by zero and ensures that scenarios without fallback nodes are appropriately considered.

Action Granularity

Action granularity measures the degree to which actions are decomposed into smaller sub-actions or components. The relative nature of Action Granularity allows for its application in a comparative manner, enabling the assessment of multiple BTs to determine which one exhibits a higher level of action decomposition. In a more granular BT, multiple action nodes can be combined to create a single action node in a less granular BT. Granularity can be compared by merging multiple action nodes into a single node and verifying whether pre-conditions and effects remain unchanged.

4.3 Properties

This section provides a comprehensive overview of the properties commonly used in the BT community, along with an analysis of their interrelations. Similar to the classification applied to metrics, BT properties are categorized into *Functional* and *Non-Functional* properties. Additionally, the relationship between these properties and evaluation metrics is analyzed, distinguishing between General (G) and Application-Specific (AS) metrics. General metrics apply universally to a given property, while Application-Specific metrics are relevant only in particular contexts.

Table 4.1 summarizes these connections, serving as a key reference for selecting the most appropriate metrics for assessing each property, thus facilitating a systematic approach to their evaluation.

4.3.1 Functional Properties

Functional properties refer to the ability to achieve specific goals that depend on the BT execution.

Reactivity

Colledanchise & Ögren [29] describe reactivity as the ability of a system to respond quickly and efficiently to changes or perturbations in its environment. This means that when an unexpected change occurs, the BT halts its current execution to react accordingly. Based on the broader concept of reactivity in control architectures, Biggar et al. [12] formally define reactivity as follows:

Definition 4.3.1 (Reactivity). A BT is reactive if its decision-making depends *only* on the current state of the environment, regardless of the execution of the correct action.

Importantly, reactivity pertains solely to the ability to respond to the change itself and does not encompass the correctness of the subsequent action

taken, which instead relates to correctness and robustness (see Definitions 4.3.4 and 4.3.5).

In general, reactivity is an intrinsic property of BTs due to their repeated ticking mechanism, which continuously polls the environment for updated information. However, the structure and design of a BT significantly influence its level of reactivity.

Evaluation As shown in Table 4.1, **execution time** (G) quantifies a BT’s responsiveness by measuring the delay in reacting to perturbations and initiating actions. A shorter execution time indicates higher reactivity, as demonstrated by Neufeld et al. [120], who compared BTs and HTN planners in a planning context.

Condition Check Frequency (G) tracks how often Condition Nodes are ticked, reflecting how frequently the BT updates decisions based on environmental changes. A higher frequency suggests improved real-time adaptability.

Since robustness combines reactivity and correctness (see Definition 4.3.5), **Behavior Usage Accuracy** (AS) and **Desired Behavior Success Rate** (AS) can be used when response time is not the focus. Cáceres Domínguez et al. [38] use the latter to assess reactivity by introducing artificial disturbances in a pick-place-push task. However, these metrics do not differentiate between a reactive but incorrect response.

Single Node Success Rate (AS) can be particularly relevant when a BT includes a critical condition node requiring an immediate response. It evaluates whether the BT reacts appropriately to specific environmental triggers, making it a valuable application-specific metric for assessing reactivity in crucial decision-making scenarios.

Efficiency

In the BT community, efficiency is often associated with time efficiency [27]. However, a broader definition is adopted in this work, aligning with the interpretation commonly used in software engineering [1], which encompasses multiple types of resource consumption.

Definition 4.3.2 (Efficiency). The degree to which a Behavior Tree performs its designated behavior with minimum consumption of resources.

Evaluation As shown in Table 4.1, **Use of Resources** (G) is the primary metric for assessing efficiency, measuring how well a BT executes its intended behavior while minimizing resource consumption. Since efficiency depends on the specific resource evaluated, its interpretation varies by application. In many domains, such as swarm robotics [74] and robot manipulation [142, 100], execution time (measured in seconds) is the most common efficiency indicator.

Predictability Distance (AS) can be useful, as higher values often correlate with longer execution times, making it relevant for comparing BT efficiency in certain contexts.

Non-functional metrics, such as **Tree Size** [19] and **Condition-to-Fallback Ratio** [65], also aid in efficiency evaluation. While these metrics do not directly measure BT performance, they serve as design heuristics, guiding the development of more efficient structures for specific applications.

Safety

The concept of safety in BTs is defined by Colledanchise & Ögren [29] as:

Definition 4.3.3 (Safety). The ability to avoid specific parts of the state space that can result in irreversible undesired behaviors.

Ensuring safety in BTs is critical for autonomous systems to operate reliably while avoiding catastrophic outcomes. A BT is considered safe if:

1. All action nodes are safe. That is, if the state $s(t)$ is safe and we execute action a , we are guaranteed to transition to $s(t+1)$ that is safe. Actions can be considered safe, given a set of pre-conditions $p(s(t))$ hold.
2. The BT design guarantees that all actions a_i are always executed only in states that satisfy their pre-conditions.

Hence, the specification of relevant conditions for each action plays a crucial role in reinforcing the above considerations.

Evaluation Safety is primarily assessed using the **Unsafe State Count** (G) and **Desired Behavior Success Rate** (G) (Table 4.1). The former evaluates a BT’s ability to maintain safety in scenarios prone to unsafe states, such as potential human collisions. Colledanchise et al. [23] assess safety by introducing artificial faults (e.g., low battery levels) to observe system responses in critical situations. While these metrics do not guarantee absolute safety, they provide valuable insights into BT reliability under testing conditions.

In application specific contexts, **Single Node Success Rate** (AS) assesses compliance with safety constraints when a BT includes condition nodes enforcing such requirements. Similarly, **Accuracy for Behavior Usage** (AS) helps assess whether sub-behaviors are expected to adhere to safe execution patterns.

Correctness and Robustness

In computer science, a program is considered correct if it is “free from faults in its specification, design, and implementation”[1]. Following the definition by Manna [107], correctness in BTs is defined as:

Definition 4.3.4 (Correctness). The correctness of a BT refers to its ability to exhibit the intended behavior throughout the entire execution.

A BT is considered correct if its execution accurately reflects the intended behavior, not only in terms of the final outcome but also in how intermediate steps are carried out. This includes adherence to the correct sequencing and timing of actions, as well as compliance with any task-specific constraints or requirements.

In dynamic environments, ensuring correctness under varying conditions is essential. As a result, the concept of robustness is introduced as a measure of how well a BT maintains its correctness despite environmental changes.

Definition 4.3.5 (Robustness). A BT is *robust* if it is correct in the presence of environmental changes.

The environmental changes considered in this definition can be categorized into two types: variations in the task domain, such as differences in object positions or sizes in a pick-and-place scenario (large domain inputs), and modifications that occur while the BT is already in execution (changes during BT execution). A BT is considered robust if it maintains its correct behavior under at least one of these conditions.

The second type of change aligns with the concept of reactivity (see Definition 4.3.1), as both refer to handling runtime changes. However, robustness extends beyond mere reactivity; it also requires that the BT executes the correct action in response to the environmental shift. For this reason, robustness can be understood as a combination of correctness and reactivity.

In general, the more failure scenarios a BT accounts for, the more robust it becomes.

Evaluation Correctness and robustness are primarily assessed using **Desired Behavior Success Rate**(G) and **Accuracy for Behavior Usage**(G) (Table 4.1). For correctness, a single evaluation run may suffice in deterministic scenarios to verify BT behavior [82, 172].

Robustness is typically evaluated by testing the BT under varying task conditions. Large domain inputs can be introduced to assess whether the BT maintains correct execution across different configurations [61]. Additionally, artificial perturbations during execution help determine the BT’s ability to adapt and sustain correct behavior under runtime changes [176, 38].

While **Graph Edit Distance** (AS) has been used to evaluate correctness [171], as stated in the previous chapter, its reliability is questionable due to its emphasis on structural similarity while disregarding BT semantics. Since semantic integrity is crucial for assessing correctness and robustness, this author does not recommend GED as a definitive evaluation metric.

4.3.2 Non Functional Properties

Non functional properties are not related to the BT functionality, and they can be assessed without the need for BT execution.

Modularity

In line with the definition of modularity provided in the IEEE Standard Glossary of Software Engineering Terminology [1], modularity in BTs is defined as:

Definition 4.3.6 (Modularity). The degree to which a Behavior Tree is composed of decoupled sub-trees, such that a change to one sub-tree has minimal impact on the others.

This definition aligns with that of Colledanchise & Ögren [29], who emphasize that independent sub-trees facilitate the separation and recombination of modular components. Modularity is a property that intrinsically characterizes BTs, as each sub-tree forms an independent module connected through the common standard interface of return statuses.

The modularity of a BT is closely related to the granularity of its actions, which refers to the level of detail in defining or decomposing behaviors. A higher level of granularity enhances modularity by increasing flexibility and reusability, allowing individual sub-actions to be combined and rearranged into different behaviors. Conversely, lower granularity reduces modularity by tightly coupling actions, making them harder to separate or modify.

Evaluation Action Granularity (G) measures modularity by assessing the decomposition of actions within a BT. Higher granularity indicates greater modularity and structural flexibility.

The *total number of nodes* (AS) can be used as an indicator of modularity, with higher values suggesting a more modular, component-based structure. However, this metric alone is insufficient, as modularity also depends on how behaviors are organized and encapsulated within sub-trees.

Graph Edit Distance (AS) can provide additional insights by evaluating a BT's adaptability to structural modifications [78].

Interpretability

The concept of interpretability in BTs was first introduced by Wathieu et al. [171] based on Post Hoc Interpretation [119]. However, no formal definition of interpretability for BTs had been established prior to this. To address this, interpretability is defined as follows:

Definition 4.3.7 (Interpretability). The degree to which the design of the BT conveys its expected behavior.

A highly interpretable BT enables users to understand its structure and predict the agent’s responses to various inputs without requiring prior knowledge of its internal mechanics. In previous literature [47, 155, 85, 128], related terms such as transparency and readability have also been used to describe this property.

Evaluation Measuring interpretability remains a challenge across various fields. Siu et al. [157] conducted a user study to analyze how individuals comprehend formal specifications, while Pattaraporn et al. [165] evaluated a BT-based robot behavior creation system for non-expert users. In alignment with these studies, interpretability in BTs is best assessed through *user studies*, as indicated in Table 4.1. Collecting user feedback provides valuable insights into design choices that enhance interpretability.

Table 4.1: Metrics and properties are classified as functional (F) or non-functional (NF), following the definitions provided in Section 4.2 for metrics and Section 4.3 for properties, respectively. Each metric-property pair is classified as General (Green) if the metric serves as a foundational benchmark for assessing that property across diverse contexts. Application-specific (Blue) metric-property pairs are tailored to a particular application scenario. Selecting and applying them requires careful consideration of the specific requirements and details of the property being evaluated. Non-Recommended (Gray) metric-property pairs indicate potential limitations or drawbacks that practitioners should carefully consider if they want to apply the particular metric to the task of measuring the property in question. Additionally, the table links each metric-property relationship to relevant works in the state of the art, providing references that support their application in BT evaluation.

	F					NF	
	Reactivity	Efficiency	Safety	Correctness	Robustness	Modularity	Interpretability
Use of Resources	* [120]	[142, 68, 106, 74, 100, 164, 138, 48, 161, 158, 99, 145, 139, 174, 181, 180, 179, 46, 36, 129]					
Predictability Distance				[142]			
Accuracy for Behavior Usage							
Single Node Success Rate							
Desired Behavior Success Rate	[88, 129]		[23, 20]	[110, 138, 94, 49, 172, 82, 20]	[88, 127, 150, 176, 61, 138, 161, 181, 129, 120, 48, 113]		
Unsafe State Count							
Condition Check Frequency							
Tree Dimensions		[19, 143]					[124, 171, 144]
GED				[171]		[78]	
Condition-to-Fallback Ratio		[65]					
Action Granularity							
User Study							

* refers to time

4.3.3 Design Recommendations

Effective BT design requires a structured approach to optimize performance across various properties. Table 4.2 outlines key design principles that enhance these properties, providing a practical reference for BT designers.

Table 4.2: Design recommendations for BT properties

Property	Design Recommendations
Reactivity	<ul style="list-style-type: none"> - Place critical conditions near the root to reduce response time. - Minimize the number of nodes. - Use parallel nodes for concurrent execution of multiple behaviors, provided they do not conflict.
Efficiency	<ul style="list-style-type: none"> - Minimize unnecessary condition checks to reduce computation overhead. - Optimize tree structure to ensure only necessary evaluations occur. - Limit excessive parallel execution to prevent resource contention.
Safety	<ul style="list-style-type: none"> - Place safety-critical checks near the root to prevent hazardous actions. - Ensure explicit precondition verification before executing risky behaviors. - Increase reactivity to quickly adapt to potentially unsafe scenarios.
Correctness	<ul style="list-style-type: none"> - Ensure the BT follows task constraints, execution order, and safety protocols.
Robustness	<ul style="list-style-type: none"> - Improve reactivity to enhance adaptability under environmental variations, even when not all variations are known in advance
Modularity	<ul style="list-style-type: none"> - Decompose actions into independent sub-behaviors (sub-trees). - Ensure each sub-tree contains only necessary conditions.
Interpretability	<ul style="list-style-type: none"> - Clearly define conditions and actions to improve clarity. - Organize BTs into a compact, well-structured format.

4.3.4 Interrelationship among Properties

Each property plays a significant role in shaping the behavior of the robot and its performance. However, optimizing one property can potentially compromise another. By gaining a better understanding of how modifications in one property can influence the others, designers can make well-informed decisions that align with the specific requirements and objectives of a given system.

Properties can often involve trade-offs that need to be carefully considered during the design phase. One instance of this is the *efficiency-reactivity trade-off*,

Another relevant trade-off is the efficiency-correctness balance, referred to as *effectiveness*. While effectiveness has been acknowledged in the literature by Cai et al. [19], no precise definition has been established. The author defines it as the *degree to which a BT is both correct and efficient*. In practical scenarios, optimizing the execution of a BT by skipping certain evaluations may lead to erroneous outcomes or failures to fulfill the intended behaviors. On the other hand, prioritizing *correctness* without considering *efficiency* can result in unnecessarily complex or time-consuming tree evaluations.

A similar trade-off exists between the number of nodes and correctness, leading to the concept of *redundancy* [150, 30]. Redundancy refers to the *degree to which each sub-tree within the BT is indispensable for achieving the intended behavior*. Evaluating redundancy helps determine whether all subtrees are necessary or if certain elements can be removed without affecting correctness. Additionally, redundancy influences interpretability. The presence of unnecessary nodes increases complexity, making the BT harder to understand and negatively impacting interpretability.

Accounting for these interdependencies is critical to aligning BT design with system-specific goals. A well-balanced approach ensures that the resulting BT structure remains both functional and optimized for the intended application.

4.4 User Study on Interpretability

Understanding and interpreting the behaviour of autonomous systems is crucial for enabling transparency, trust, and human oversight. In this context, BTs are widely used due to their modular structure and intuitive control flow semantics. However, as BTs become more complex, their interpretability by human users may degrade, particularly when non-trivial design patterns and control structures are involved.

To systematically investigate the factors that affect the interpretability of BTs, a controlled user study was designed. The study explores how structural complexity, design patterns, and prior familiarity with BTs influence human comprehension and confidence when reasoning about tree execution. The study

was conducted in three main phases: a pre-study with BT experts to ground the experimental design, the construction and deployment of a Qualtrics¹-based survey, and subsequent data analysis to evaluate a series of predefined hypotheses.

4.4.1 Experimental Protocol

The experimental protocol consisted of two stages: a qualitative pre-study to identify common BT design patterns, and the creation of a structured survey to assess interpretability under controlled conditions.

Pre-study

In order to identify common BT design patterns used in practice, a pre-study involving six BT experts recruited via academic and industry networks was conducted. Each expert was asked to construct BTs for three out of six predefined scenarios. These scenarios were accompanied by structured materials: (1) a textual description of the scenario, (2) a list of permitted action nodes, (3) a list of available condition nodes, and (4) the set of usable control flow nodes: Fallback, Sequence, Parallel, and Inverter. Experts were instructed to model the behavior using only the provided components and were free to use any graphical tool or to draw by hand.

The scenarios were designed to balance structural complexity and control flow requirements. Two were intentionally simple (fewer than 4 action and 4 condition nodes), two were more complex (7–9 action nodes and up to 10 conditions), and two were constructed to require inherently parallel behavior, while maintaining a small number of nodes. The six scenarios were evenly distributed across three expert groups, ensuring three independent solutions per scenario.

Following submission, each expert participated in a 30-minute semi-structured interview to discuss their design rationale.

BT Design Patterns

Based on this qualitative analysis of the pre-study and prior work on BT modelling idioms [39], four dominant BT design patterns were identified. Each design pattern aims to highlight a specific modelling strategy.

Fallback-on-top. This design pattern places a fallback node at the root, whose branches represent mutually exclusive behaviors or recovery strategies. Each action is typically guarded by a sequence node containing its preconditions. The overall logic can be expressed as “if precondition, then do action”.

¹<https://www.qualtrics.com/>

This idiom corresponds to the Robust Logical-Dynamical Chain (RLDC) pattern [39] and in this formulation:

1. Actions are listed in reverse execution order, with the final action appearing first and the initial action last
2. The root is a fallback node.
3. Each action is guarded by a precondition rooted in a sequence.

A representative example of this design is shown in Figure 4.1a.

Sequence-on-top. This design pattern places a Sequence node at the root and encodes a linear arrangement of actions, each guarded by a conditional check. Specifically, each child of the Sequence is a Fallback node that first checks a negated precondition. If the negated precondition fails (i.e., the actual precondition is true), the Fallback proceeds to execute the corresponding action. In other words, the action is only executed when its precondition holds. In this formulation

1. The root node is a Sequence
2. Each child is a Fallback node that first evaluates the negated precondition of the action; if this check fails (i.e., the actual precondition holds), the corresponding action is executed.

A representative BT structure is shown in Figure 4.1b.

PPA (Precondition–Postcondition–Action) Patterns. The PPA pattern, initially formalized by Colledanchise & Ögren [29], groups actions that share a common postcondition but differ in their preconditions. Structurally, a fallback node is used to coordinate alternative actions; each branch is a sequence comprising a precondition check and the associated action. This approach improves robustness by allowing multiple ways to achieve the same goal and is widely used in BT-based planning systems.

Three variants of this pattern were used in the study:

- **PPA:** A fallback node is rooted on the desired postcondition. Each branch checks a specific precondition before attempting the associated action. This form matches the structure described above and is exemplified in Figure 4.1c.
- **PPA Backward:** This variant applies the principles of backward chaining [22, 19] to BT construction. Starting from a high-level goal, the tree recursively expands preconditions into nested PPA subtrees. This results in a layered structure in which each precondition is backed by an alternative plan to achieve it. An illustration is provided in Figure 4.1d.

- **PPA-split:** A composite variant combining modularity and backward chaining. The tree is rooted in a fallback node representing the final post-condition. Each branch corresponds to a subgoal and is itself structured as a PPA Backward subtree. This design enables hierarchical refinement while maintaining interpretability. See Figure 4.1e for an example.

Parallel-on-top. Parallel-on-top pattern was included because it is used in existing tools such as RE:BT Espresso [47]. Two variants were used:

- **Parallel Sequence:** A parallel node where all children must succeed (i.e., success threshold equals the number of branches). This structure is suitable when multiple independent tasks must be completed jointly (e.g., navigating while monitoring sensors). Figure 4.1f shows an example.
- **Parallel Fallback:** A parallel node configured to succeed if any branch succeeds. This form captures opportunistic execution or distributed strategies. An example is provided in Figure 4.1g.

Another modeling pattern identified across all expert participants in the pre-study was the handling of inverter nodes. Rather than employing inverter nodes, most experts preferred to encode negation directly within the condition node names (e.g., *not detected*, *battery not full*). This practice is believed to enhance readability by simplifying tree traversal and reducing the cognitive effort required for interpretation.

Survey creation

Based on the design patterns identified in the pre-study, a structured online survey was developed to evaluate BT interpretability across a range of conditions. Participants first answered demographic questions (e.g., education level, BT familiarity) and completed a brief tutorial on BTs. They were then presented with eight multiple-choice questions. Each question consisted of a visual representation of a BT, a textual prompt (e.g., “Given that node X just succeeded/failed, which node will be executed next?”), and a set of candidate answers. After each answer, participants rated their confidence from 1 to 7, 1 being “This was a guess”, and 7 being “I am completely sure”.

The BTs presented in the survey were designed based on four scenario categories: easy, medium, difficult, and parallel. The categorization into easy, medium, and difficult was determined by structural complexity, defined in terms of the number of actions: easy scenarios included 1–2 actions, medium scenarios 3–4 actions, and difficult scenarios more than 4 actions. These thresholds were defined based on the results presented by Eriksson [42]. In the present study, two easy scenarios with 2 actions, two medium scenarios with 4 actions, and two difficult scenarios with 9 actions were used.

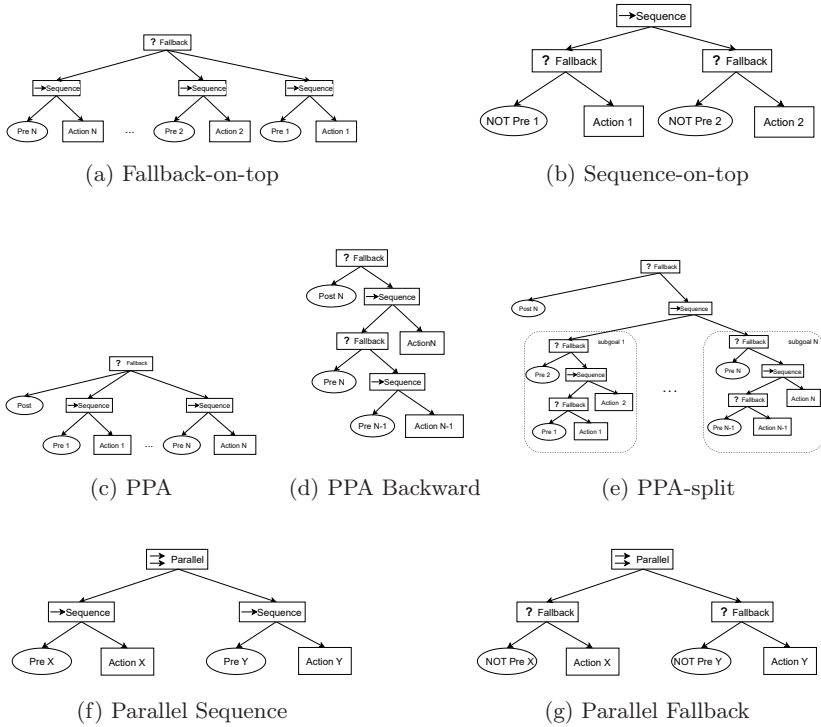


Figure 4.1: Overview of BT design pattern examples.

In contrast, parallel scenarios were not defined by action node counts but rather by their behavioural semantics. These scenarios featured inherently parallel tasks, such as concurrent movement and perception, and were designed to assess interpretability when multiple execution paths are active by default. An example includes the Pac-Man scenario, where the agent must simultaneously check its environment and navigate toward objectives. This type of parallelism introduces unique interpretive challenges, distinct from purely sequential task structures.

Each participant was randomly presented with one BT per scenario, resulting in eight unique trials per individual. The randomization ensured that the design patterns were balanced across participants. However, due to limitations of Qualtrics, it was not possible to also randomize the order of questions. Therefore, participants were presented with the same order of difficulty, namely “easy”, “medium”, “hard”, “hard”, “easy”, “medium”, “parallel”, “parallel”. We tested for learning effect (see 4.4.4) before performing the analysis.

In contrast, parallel scenarios were not defined by node counts but rather by their behavioural semantics. These scenarios featured inherently parallel tasks, such as concurrent movement and perception, and were designed to assess interpretability when multiple execution paths are active by default. An example includes the Pac-Man scenario, where the agent must simultaneously check its environment and navigate toward objectives. This type of parallelism introduces unique interpretive challenges, distinct from purely sequential task structures.

To comprehensively evaluate interpretability across different types of BT structures six BT designs were constructed per scenario, including Fallback-on-top, Sequence-on-top, PPA, PPA Backward, Parallel Sequence, and Parallel Fallback architectures. For difficult scenarios, an additional Split PPA design was incorporated to accommodate increased structural complexity.

4.4.2 Hypothesis

Based on the pre-study, the following hypotheses were formulated:

- H1 The size of the tree negatively impacts interpretability.
- H2 The type of design impacts interpretability, and ppa split design is easier to interpret.
- H3 Wide trees are more interpretable than deep trees.
- H4 Familiarity with behavior trees positively impacts interpretability.

Since interpretability cannot be directly observed, it was assessed through two proxies: response correctness and confidence. Prior work shows that interpretability is typically measured indirectly, e.g., by users' ability to predict system behavior [40], or via task performance and self-reported confidence [76]. In this study, correctness is a binary variable indicating whether each response was answered correctly, while confidence is an aggregated measure of participants' confidence in the correctness of their answer.

4.4.3 Sample

Participants were recruited through online dissemination channels. No prior knowledge of BTs was required. A total of 235 participants accessed the online survey. After applying exclusion criteria, the final dataset consisted of 86 valid and complete responses. Exclusion was based on three criteria: (1) survey completion status, (2) minimum engagement time (responses completed in under three minutes were discarded), and (3) entries flagged as fraudulent or duplicated by Qualtrics' built-in detection mechanisms.

4.4.4 Preliminary Analysis

Familiarity with behavior trees

The distribution of self-reported familiarity with behavior trees is presented in Table 4.3. Given the very low number of participants who had never encountered behavior trees, the categories “Never heard of them” and “I have heard of them but I don’t know much” were aggregated into a single “Low Familiarity” category. The resulting variable, BT Familiarity, is shown in Table 4.4.

Table 4.3: Distribution of answers for familiarity with BTs

BT Familiarity	# of Answers	Percentage
Never heard of them	3	3.5
I heard of them but I don’t know much	17	19.8
Somewhat familiar	37	43.0
Very familiar	29	33.7

Table 4.4: Aggregated BT familiarity variable

BT Familiarity Level	# of Answers	Percentage
Low Familiarity	20	23.3
Medium Familiarity	37	43.0
High Familiarity	29	33.7

Learning Effect

Due to the limitations of the tools used for the survey, it was not possible to randomize the order of the questions, resulting in all participants responding to the difficulty categories in the same sequence. Prior to conducting the main analysis, a test for potential learning effects was carried out. Specifically, a McNemar test for paired samples was applied to each difficulty category. The results indicated no significant differences across the groups ($p_{easy} = 0.105$, $p_{medium} = 0.194$, $p_{hard} = 0.063$, $p_{parallel} = 0.622$). On this basis, it was concluded that no learning effect was present, and each response was subsequently treated as an independent data point in the remainder of the analysis.

Correlation Between Confidence Level and Response Correctness

Although confidence level and response correctness were used as proxy measures of interpretability, a correlation between them was suspected and was therefore examined, specifically with familiarity with BTs as a mediator variable. Pearson’s Chi-Square tests were performed between confidence level and response correctness, layered by familiarity with BT. Overall, a correlation

between confidence level and response correctness was observed ($\chi^2 = 71.04$, $p < .001$) (Table 4.5).

Further analysis explored whether this correlation was moderated by participants' self-reported familiarity with behavior trees. Among participants with low familiarity, a significant correlation was observed ($\chi^2 = 14.81$, $p < .001$), with notable deviations for low confidence and incorrect answers ($std = 3.52$) and high confidence with correct answers ($std = 2.86$) (Table 4.6). A similar pattern emerged for participants with medium familiarity ($\chi^2 = 36.18$, $p < .001$), where low confidence and wrong answers ($std = 5.02$) and high confidence with correct answers ($std = 5.39$) (Table 4.7).

Table 4.5: Chi-Square test for the correlation Confidence - Response Correctness. Significant p-value are highlighted in **bold**

BT Familiarity		Value	df	p
Low	X ²	14.812	2	< .001
	N	160		
Medium	X ²	36.177	2	< .001
	N	296		
High	X ²	4.186	2	0.123
	N	232		
Total	X ²	71.047	2	< .001
	N	688		

Table 4.6: Contingency table - Confidence / Response Correctness for low familiarity. Significant deviations are highlighted in **bold**

Confidence Level		Correct Answer		Total
		false	true	
Low	Count	33	21	54
	Standardized residuals	3.520	-3.520	
Medium	Count	25	41	66
	Standardized residuals	-0.859	0.859	
High	Count	9	31	40
	Standardized residuals	-2.868	2.868	
Total Count		67	93	160

Table 4.7: Contingency table - Confidence / Response Correctness for medium familiarity. Significant deviations are highlighted in **bold**

Confidence Level		Correct Answer		Total
		false	true	
Low	Count	35	17	52
	Standardized residuals	5.020	-5.020	
Medium	Count	37	46	83
	Standardized residuals	1.727	-1.727	
High	Count	37	124	161
	Standardized residuals	-5.392	5.392	
Total Count		109	187	296

However, for participants with high familiarity, no significant correlation was found ($p = 0.12$). Descriptive statistics revealed that 81% of their responses were given with high confidence (Table 4.8), suggesting a potential overconfidence effect.

Table 4.8: Reported confidence levels for participants with a high familiarity with BTs.

Confidence Level	Frequency	Percent
Low	15	6.466
Medium	28	12.069
High	189	81.466

4.4.5 Hypothesis Testing

In this section, the tests performed and the results obtained for each of the hypotheses are reported. Unless stated otherwise, all tests have been done using Pearson's Chi-Square test.

H1 - The size of the tree negatively impacts interpretability

No significant correlation was found between tree size and response correctness ($p = 0.4$), nor between tree size and confidence level ($p = 0.7$). These results suggest that overall size, in terms of number of nodes, does not directly influence interpretability in the experimental conditions tested.

H2 - The design of the tree impacts interpretability

Results showed a significant association between tree design and both response correctness ($\chi^2 = 16.2$, $p = 0.04$) and confidence ($\chi^2 = 16.8$, $p = 0.02$). Analysis of the residuals showed a strong deviation of the expected frequencies in

the parallel fallback and wrong answer ($std = 2.60$) as well as in the PPA-split and correct answer ($std = 2.51$) (Table 4.9). For the confidence level, analysis of the residuals revealed significant deviations from the expected frequencies in several cases (see Table 4.10). Specifically, deviations were observed for parallel fallback with low confidence ($std = 3.60$) and with high confidence ($std = -2.26$), as well as for parallel sequence with low confidence ($std = 2.47$) and with high confidence ($std = -2.09$). These findings indicate that the parallel fallback design is associated with fewer correct answers and lower confidence, whereas the parallel sequence design is characterized by lower confidence despite the frequency of correct answers aligning with expectations. In contrast, the ppa-split design shows a higher frequency of correct answers while confidence levels remain consistent with the expected distribution.

Table 4.9: Contingency table - Design / Response Correctness. Significant deviations are highlighted in **bold**.

Design		CorrectAnswer		Total
		false	true	
fallback on top	Count	43	90	133
	Standardized residuals	-0.181	0.181	
parallel fallback	Count	39	47	86
	Standardized residuals	2.605	-2.605	
parallel sequence	Count	28	43	71
	Standardized residuals	1.219	-1.219	
ppa	Count	25	65	90
	Standardized residuals	-1.129	1.129	
ppa split	Count	1	17	18
	Standardized residuals	-2.509	2.509	
ppa+backward	Count	21	54	75
	Standardized residuals	-0.975	0.975	
pure parallel fallback	Count	21	36	57
	Standardized residuals	0.645	-0.645	
pure parallel sequence	Count	17	34	51
	Standardized residuals	0.054	-0.054	
sequence on top	Count	32	75	107
	Standardized residuals	-0.739	0.739	
Total Count		227	461	688

Table 4.10: Contingency table - Design / Confidence. Significant deviations are highlighted in **bold**.

Design		ConfidenceLevel			Total
		Low	Medium	High	
fallback on top	Count	17	40	76	133
	Standardized residuals	-1.621	1.277	0.118	
parallel fallback	Count	27	20	39	86
	Standardized residuals	3.596	-0.560	-2.268	
parallel sequence	Count	20	19	32	71
	Standardized residuals	2.473	0.210	-2.086	
ppa	Count	12	20	58	90
	Standardized residuals	-1.137	-0.816	1.593	
ppa split	Count	1	5	12	18
	Standardized residuals	-1.359	0.202	0.866	
ppa+backward	Count	8	23	44	75
	Standardized residuals	-1.668	1.037	0.367	
pure parallel fallback	Count	11	17	29	57
	Standardized residuals	0.354	0.739	-0.924	
pure parallel sequence	Count	6	11	34	51
	Standardized residuals	-1.135	-0.706	1.495	
sequence on top	Count	19	22	66	107
	Standardized residuals	0.050	-1.330	1.135	
Total Count		121	177	390	688

H3 - Wide trees are more interpretable than deep trees

The width-to-height ratio was calculated for each of the designed trees and subsequently categorized according to Table 4.11.

Table 4.11: Categorization of the width-to-height ratio

Ratio Category	Width-to-height Ratio	Number of trees
Deep	$0 \leq r < 1$	6
Balanced	$1 \leq r < 2$	12
Wide	$2 \leq r < 6$	16
Extra-Wide	$6 \leq r$	10

Results showed a significant correlation ($chi = 11.8$, $p = 0.0081$) between the ratio category and response correctness. The analysis of residuals (see Table 4.12) showed a significant deviation from expected frequencies for balanced trees ($std = 2.44$), where the frequency of correct answers is higher than expected, and extra-wide trees ($std = 2.82$), where the frequency of correct

answers is lower than expected. However, there was no significant correlation between the ratio category and the confidence level ($p = 0.07$).

Table 4.12: Contingency table - Ratio category / Response Correctness. Significant deviations are highlighted in **bold**

Ratio Category		CorrectAnswer		Total
		false	true	
Deep	Count	21	54	75
	Standardized residuals	-0.975	0.975	
Balanced	Count	52	147	199
	Standardized residuals	-2.443	2.443	
Wide	Count	99	189	288
	Standardized residuals	0.654	-0.654	
Extra-wide	Count	55	71	126
	Standardized residuals	2.815	-2.815	
Total Count		227	461	688

H4 - Familiarity with behavior trees positively impacts interpretability

Results showed a significant correlation ($chi = 20.4$, $p < 0.001$) between the self-reported familiarity with BTs and response correctness. The analysis of residuals (see Table 4.13) showed significant deviations in low familiarity ($std = 2.73$), where the frequency of correct answers is lower than expected, and in high familiarity ($std = 4.38$), where the frequency of correct answers is higher than expected.

Table 4.13: Contingency table - Familiarity with BT / Response Correctness. Significant deviations are highlighted in **bold**

Familiarity with BT		CorrectAnswer		Total
		false	true	
Low	Count	67	93	160
	Standardized residuals	2.727	-2.727	
Medium	Count	109	187	296
	Standardized residuals	1.857	-1.857	
High	Count	51	181	232
	Standardized residuals	-4.382	4.382	
Total Count		227	461	688

Results also showed a significant correlation ($chi = 126.25$, $p < 0.001$) between self-reported familiarity with BTs and confidence level. The analysis

of residuals (see Table 4.14) showed significant deviations in the low familiarity group, where the frequency of high confidence scores was significantly lower than expected ($std = -9.23$), and in the high familiarity group, where the frequency of high confidence scores was significantly higher than expected ($std = 9.36$).

Table 4.14: Contingency table - Familiarity with BT / Confidence. Significant deviations are highlighted in **bold**

Familiarity with BT		ConfidenceLevel			Total
		Low	Medium	High	
Low	Count	54	66	40	160
	Standardized residuals	6.130	5.128	-9.233	
Medium	Count	52	83	161	296
	Standardized residuals	-0.012	1.206	-1.055	
High	Count	15	28	189	232
	Standardized residuals	-5.466	-5.846	9.356	
Total Count		121	177	390	688

4.4.6 Discussion of the Findings

The results of the user study provide insight into how structural properties of BTs, design patterns, and individual familiarity influence interpretability, measured through response correctness and confidence. The following discussion addresses each hypothesis in turn and interprets the observed effects in the context of human understanding and system design.

H1 - Number of nodes. Contrary to expectations, tree size, measured by node count, did not significantly affect interpretability or confidence. This finding suggests that overall size may be less critical than other structural factors, such as layout or modularity. This result also indicates that visual complexity alone does not necessarily lead to lower interpretability, provided the underlying structure remains logically coherent.

H2 - Type of design. The results provide partial support for the claim that tree design impacts interpretability. Specifically, trees based on the PPA-split pattern yielded higher accuracy, suggesting that modular designs with explicit goal decompositions support user reasoning. Conversely, parallel fallback trees were associated with higher error rates, potentially due to their semantic ambiguity or non-intuitive control flow. The case of parallel sequence trees is interesting, as it was not associated with better accuracy, but with higher confidence. This indicates that this design may seem more understandable by

users, indicating a decoupling in perceived and actual complexity. These findings reinforce the notion that not all BT idioms are equally interpretable, and that certain design patterns, especially those grounded in goal structure, are better suited for human-in-the-loop systems.

H3 - Tree shape. The results provide partial support that the tree shape, measured by the width-to-height ratio, impacts interpretability, but exclusively in the response correctness metric. Specifically, balanced trees (ratio between 1–2) were interpreted more accurately than wide (2–6) or extra wide (above 6) configurations. However, while extremely wide trees were associated with lower accuracy, deep trees (ratio below 1) did not show any significant association with accuracy.

No effect of this ratio on confidence was detected, indicating a possible decoupling between perceived and actual complexity in this case as well. Participants did not report higher confidence when facing wider trees, even when performance degraded, suggesting that users may not consciously perceive increased width as more difficult, even though it negatively impacts their performance. This discrepancy between subjective perception and objective difficulty merits further investigation, especially for tools that automatically generate or visualize BTs.

H4 - Familiarity and interpretability. The analysis provided strong evidence that familiarity with BTs influences interpretability. Interestingly, the preliminary analysis showed that among highly familiar users, confidence remained uniformly high regardless of response correctness, revealing a pattern of overestimation. In contrast, participants with low or medium familiarity exhibited a meaningful alignment between their confidence ratings and performance. This dissociation in the high-familiarity group suggests that domain expertise may introduce a bias in self-assessment, potentially reducing vigilance during error-prone tasks—a critical consideration for interface and feedback design in expert systems.

Taken together, the findings emphasize that interpretability is not determined by structural factors alone but emerges from the interaction of design, complexity, and user experience. Design patterns that align with logical planning strategies (e.g., PPA) appear to enhance human understanding, while designs that violate user expectations (e.g., fallback-heavy or extra-wide structures) reduce interpretability. Familiarity enhances performance but introduces risks related to overconfidence, particularly in expert users. These insights underscore the need for adaptive interfaces and validation tools that not only reflect structural complexity but also account for user variability and cognitive heuristics.

4.5 Summary and Outlook

This chapter addressed the critical but underexplored question of how to evaluate the quality of BTs, particularly when they are learned from execution data. While previous chapters demonstrated how BTs can be automatically generated in a modular and interpretable format, this chapter introduced a structured evaluation framework to assess the extent to which those properties are achieved in practice.

The main contributions of this chapter are threefold. First, it proposed a taxonomy of key properties relevant to BT evaluation along with a set of well-defined, quantitative metrics aligned with each property. Second, it offered design recommendations for BT construction. Third, it presented the results of a user study aimed at empirically testing the notion of interpretability.

The findings of the user study revealed that interpretability is influenced by multiple interacting factors, including a user's familiarity with BTs, BT structure, and the design pattern used. These factors significantly affect both users' confidence in understanding the tree and the accuracy of their task-related responses. These insights reinforce the claim that BTs are a promising representation for human-in-the-loop systems, but also emphasize the importance of structure-aware design and evaluation.

The next chapter turns to a different but complementary challenge: learning planning domain models directly from logs. These models form the core of automated planning systems, and the ability to learn them from data, rather than crafting them manually, extends the thesis's broader goal of deriving interpretable, verifiable system knowledge from execution traces.

Chapter 5

Learning Planning Domains

This chapter focuses on the learning of planning domain models, which capture the relationships between actions, their preconditions, and their effects. By automating the learning of planning domains, it is possible to enable more scalable and adaptable decision-making frameworks, particularly in fleet management for autonomous vehicles. The idea is to extract planning knowledge directly from experience, improving efficiency and adaptability without requiring manual intervention.

Figure 5.1 provides an overview of this transformation by depicting the available input and the desired output. The input consists of observed state transitions and executed actions, capturing how the system evolves over time. The objective is to derive a structured planning domain representation, where actions are formally defined through operator schemas that specify their preconditions and effects.

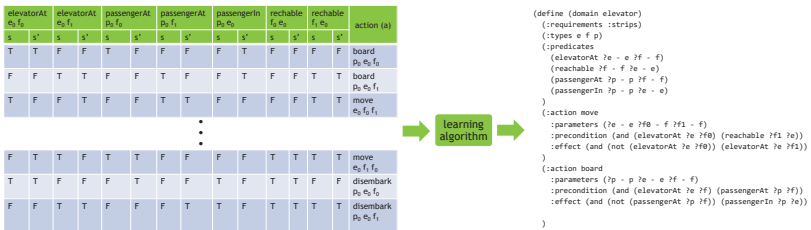


Figure 5.1: An example of the input and output in the action model learning process. The left side represents the input, consisting of structured state-action transitions, where each row captures the values of state variables before and after an action is executed. The right side illustrates the output, a learned PDDL domain representation that formalizes the action model derived from the input data.

5.1 Related Work

The field of action model learning for domain-independent planning has seen significant advancements, with various approaches developed to extract structured planning knowledge from data [7]. These approaches differ in their assumptions and methodologies.

5.1.1 Learning from Deterministic Plan Traces

Early work in action model acquisition typically assumes fully observable, noise-free state traces and relies on active experimentation or expert guidance to refine models. OBSERVER [170] alternates plan execution with model refinement to converge on correct preconditions and effects. Building on this, OpMaker [111], developed within the GIPO framework [156], introduces hierarchical operator induction and interactive assistance to help users specify more complex operators. EXPO [53] follows the same basic assumptions of OBSERVER but adds support for conditional effects, allowing richer action descriptions. TRAIL [11] employs inductive logic programming to learn preconditions from fully observed state transitions.

A second family of methods dispenses with explicit state observations entirely, treating the execution log as a labeled directed graph whose nodes correspond to abstract states. Among these, the LOCM family of algorithms - LOCM [34, 35] and LOCM2 [33] - represent actions as parameterized finite state machines. A distinctive feature of these approaches is their ability to infer state variables without prior knowledge or observable states, relying on logical induction rather than predefined predicates. However, a key limitation is that they primarily focus on dynamic properties, neglecting static relationships between objects, such as which places are connected. This limitation becomes particularly problematic considering that numerous systems rely on static predicates to define the applicability of actions within the domain.

To address this limitation, NLOCM [60] extends LOCM by inducing static predicates through a post-processing step. However, this requires additional structured input, including optimal plans and suboptimal fully observable action sequences. In real-world scenarios, obtaining fully observable traces can be impractical, making this approach less suitable for environments where only partial information is available.

Moving beyond purely symbolic or graph-based induction, optimisation-driven learners tolerate partial observability by casting model learning as a MAX-SAT problem. ARMS [178] reconstructs missing fluents by choosing preconditions and effects that minimize a weighted combination of errors (failing to explain observed transitions) and redundancies (unnecessary conditions). LAMP [187] extends ARMS to richer, ADL-style domains, supporting quantifiers and logical implications, by embedding the learning task in a Markov Logic Network. CAMA [184] instead of relying on domain-specific informa-

tion such as predicates and types, integrates crowd-sourced knowledge into the learning process. The algorithm synthesizes soft constraints from both crowd annotations and plan traces and then resolves these constraints using a weighted MAX-SAT solver. This underscores the necessity for extensive external input, either through simulated crowd-sourcing (which requires parameter tuning) or through manual labeling by human annotators. However, CAMA requires a substantial amount of data, with experiments showing strong performance only when at least 75 traces are available. An alternative to these optimization-driven approaches is SLAF [5], which takes a logic-based perspective. Rather than optimizing over weighted costs, SLAF encodes observations into a logical theory and uses constraint propagation to eliminate inconsistent models. Given partially observed states and action sequences, it incrementally refines the hypothesis space by retaining only those action models that remain logically consistent with all observations. This approach avoids the need for numerical optimization but can be computationally intensive as the number of possible models grows.

On a similar line, FAMA [4] is designed to work with plan traces where actions may be partially or fully unobservable. However, it requires an initial domain model that, while lacking preconditions and effects, includes a comprehensive set of state variables and their corresponding types, as well as complete action signatures (action names and types).

SAM [86] follows a similar “model-skeleton” philosophy, but originates from the *safe action model learning* literature [160]. Given fully observed, noise-free traces plus a domain scaffold (action names, parameter types, and predicate vocabulary), it builds a lifted action model guaranteed never to over-generalise: every learned precondition is witnessed in some trace, and every learned effect is necessary to explain an observed state change. While SAM’s safety guarantees are appealing, its reliance on a manually specified skeleton again highlights the trade-off between weaker input assumptions and the complexity of the learning task.

To further reduce input requirements, Balyo et al. [9] proposed a method for learning action schemas from state traces that include fully observable state transitions but provide no information beyond the action name. However, the approach assumes that traces are noise-free and fully observable, meaning all state changes are accurate and transitions are reliable.

Recently, several deep learning-based approaches have been explored for learning planning domains from sub-symbolic or unstructured data. Huo et al. [75] employed transformer architectures to translate natural language descriptions into planning domains for typhoon prediction. Their method takes advantage of transformers’ attention mechanisms to model long-range dependencies in sequence-to-sequence data. However, the approach is computationally demanding and depends heavily on the quality of training data, though the use of pre-trained models alleviates some of these constraints. Kase et al. [91] utilized Convolutional Neural Networks (CNNs) to generate planning

domains from image and sensor data in robotic settings. Their approach effectively captures relevant environmental features but is highly domain-specific, particularly tailored for robotic arms, and thus lacks generalizability to other domains. Xiao et al.[175] introduced a graph-based approach using Graph Neural Networks (GNNs) to learn planning domain structures from partially observed traces. This method leverages the representational power of GNNs for structured relational data but depends on the availability of graph-based representations. Iklassov & Medvedev [77] integrated GNNs with reinforcement learning to enhance planning robustness in logistics domains. While effective in complex decision-making environments, their method still requires well-formed graph-structured inputs, limiting applicability in unstructured settings. Ahmetoglu et al. [3] propose a neural-symbolic system that learns PDDL operators from continuous sensor data using a deep encoder-decoder architecture. Their model, trained on 160,000 out of 200,000 collected samples, is applied to a tabletop stacking task. While effective, the approach requires substantial training data for a single domain and relies on simulated sensory traces rather than symbolic plan traces. Another notable system is LatPlan [8], which learns grounded PDDL action models from pairs of before and after images using an unsupervised deep autoencoder framework. While it achieves good results in simple domains, its performance degrades significantly in complex scenarios due to poor image reconstruction and incorrect action modeling [101]. Additionally, LatPlan learns grounded representations, resulting in very long and complex action schemas that are difficult to interpret and scale. Moreover, because actions are defined by changes in the model’s latent space, the resulting PDDL domain is tightly coupled with the model. This limits interpretability and reusability, as plans cannot be understood outside the model and must be translated back into sub-symbolic representations for execution, complicating integration with external planning or control systems.

5.1.2 Learning with Numerical State Variables

Other approaches focus on learning planning models that include numerical state variables. N-SAM [116] extends SAM by incorporating a dedicated algorithm to handle numerical variables while maintaining the same assumptions of full, noise-free observability. PlanMiner [152], on the other hand, follows a distinct multi-step process to derive planning models from plan traces. It first converts plan traces into structured datasets, then applies symbolic regression to enrich the extracted information, and finally learns preconditions and effects by calling NSLV [57], a genetic algorithm for rule extraction. However, solving a symbolic regression problem is, in general, computationally expensive, which may limit the feasibility of this approach in real-world applications.

5.1.3 Learning from Noisy Observations

While many action model learning techniques assume clean and deterministic plan traces, several methods explicitly focus on handling noise in observed data. AMAN [185] constructs action models from noisy execution traces using graphical models, assuming that actions themselves may be affected by noise. This approach processes plan traces without intermediate states, generating a set of candidate action models that align with the observed transitions. However, it does not handle cases where intermediate states are noisy.

Building on this idea, AMDN [186] is designed to handle more complex noise patterns, including disordered actions, parallel actions, and noisy states. However, its effectiveness depends on having a large number of plan traces, with experiments using between 40 and 200 traces per scenario.

Another recent approach is AMLSI [59], which learns PDDL domain models from partial and noisy observations. AMLSI uses grammar induction over state-action sequences and constructs a deterministic finite automaton (DFA) to generalise observed behaviors. However, it requires two training sets: one containing feasible action sequences and another containing infeasible ones.

Another approach, NOLAM [99], also employs graphical models but explicitly models uncertainty in noisy states. While effective in handling noise, NOLAM tends to overconstrain action models by introducing many negative preconditions, potentially reducing the generalizability of the learned models. Unlike these probabilistic approaches, Mourão et al. [117] apply a voted Perceptron classifier to learn explicit STRIPS rules from noisy traces, training classifiers to predict action effects. However, this method requires large training sets - experiments were conducted with datasets containing 20,000 actions, making it impractical for real-world applications where only limited traces are available.

Another method for learning action models from traces with noisy states is proposed by Rovida et al. [142]. Their approach learns a set of logical rules associated with preconditions and effects of actions, then incrementally specializes or generalizes these rules as new transitions are provided. This incremental learning strategy allows the method to adapt dynamically to new data, making it particularly useful for scenarios where execution traces become available over time.

Similarly, Agravante et al. [2] propose a neuro-symbolic model that learns action models from traces with potentially noisy states. This method relies on inductive logic programming [140], modeling preconditions and effects as a set of logical rules, which are then translated into a PDDL action model. However, no experimental analysis has been conducted to evaluate the impact of noise on its performance, leaving an open question regarding its robustness in real-world noisy environments.

Finally, an extension of the PlanMiner algorithm, PlanMiner-N [151], introduces noise-handling capabilities for learning numerical planning domains.

PlanMiner-N preprocesses input data by detecting and filtering out noise before applying the standard PlanMiner pipeline. After learning preconditions and effects, it further refines the learned rules through a meta-state refinement step, identifying and correcting erroneous preconditions and effects caused by residual noise.

5.2 LAML

This section presents the Lifted Action Model Learning (LAML) algorithm, which has undergone significant evolution to improve its robustness and applicability in real-world scenarios. Initially, LAML was designed to learn action models from deterministic datasets by systematically extracting grounded preconditions and effects before generalizing them into lifted operator schemas.

In its initial formulation, LAML processes sequences of state transitions to learn a set of grounded preconditions for each grounded action using decision tree learning (C5.0 algorithm [95]). The approach consists of the following steps:

- **Learn Grounded Preconditions:** LAML applies C5.0 decision trees on each state variable to determine which grounded actions require specific conditions to hold before execution.
- **Infer Grounded Effects:** Given the deterministic nature of the environment, effects are inferred by comparing state variables between consecutive states s and s' . Any change in a state variable signifies that the action has modified it. This process is similar to inductive logic learning [118], where effects are logically derived from the action's preconditions.
- **Lifting:** The learned grounded preconditions and effects are then generalized into lifted operator schemas.
- **PDDL Conversion:** Operator schemas are used to create a Planning Domain Definition Language (PDDL) domain model.

The initial version of LAML was evaluated on benchmark domains from the International Planning Competition (IPC) and compared against state-of-the-art methods for learning planning domains in deterministic environments. The results demonstrated that LAML is a valid and effective approach (Detailed results are presented in Section 6.4.) However, this approach had notable limitations. First, its reliance on a strictly deterministic environment made action effects inference highly susceptible to noise, reducing its robustness in real-world applications. Second, because the learning process operated on grounded data, a significantly larger dataset was required to accurately infer preconditions, making the approach data-intensive and less scalable.

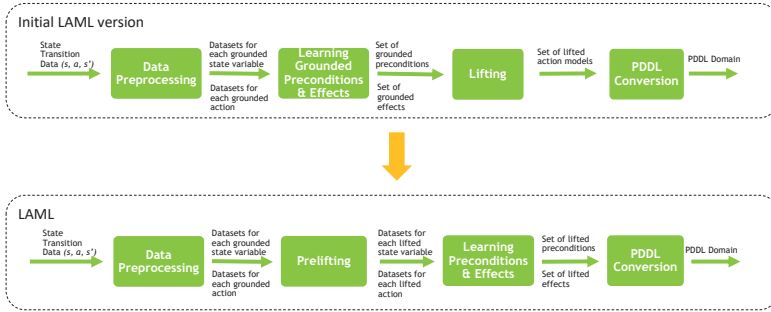


Figure 5.2: Comparison between the initial learning-then-lifting approach and the improved lifting-then-learning paradigm. In the original method, LAML first learns grounded preconditions and effects from state transitions before generalizing them into lifted operator schemas. The updated framework, however, begins by lifting the dataset, allowing for direct learning of lifted preconditions and effects.

To address these limitations, LAML was extended into a more advanced framework that incorporates dataset prelifting, allowing for direct learning of lifted preconditions and effects. Instead of first learning grounded models and then generalizing, the updated LAML pipeline first lifts the data and then applies learning techniques, significantly improving scalability. Additionally, effect learning follows a frequentist approach, making it more robust to noise by statistically identifying the most reliable effect patterns.

Figure 5.2 illustrates the transition from the initial learning-then-lifting paradigm to the improved lifting-then-learning approach.

The next section provides a detailed explanation of the final version of LAML. To illustrate key concepts, a running example based on a simplified version of the well-known elevator domain¹ will be used throughout the explanation.

Running Example: The Elevator Domain

The elevator domain models an elevator system’s operation, enabling passengers’ movement between different floors. This system is represented by a collection of objects, predicates, and actions, which delineate the potential states and changes that can occur within this domain. In detail,

- Objects: elevators (e), floors (f), passengers (p).
- State Variables (denoted as the set X):

¹Int. Planning Competition <https://ipc00.icaps-conference.org/>

- *passengerAt*(p, f): passenger p is at floor f .
 - *passengerIn*(p, e): passenger p is inside elevator e .
 - *elevatorAt*(e, f): elevator e is at floor f .
 - *reachable*(f, e): floor f is reachable by elevator e .
- Actions (denoted as the set A):
 - *board*(p, e, f): passenger p boards elevator e at floor f , requiring the passenger p to be at floor f and the elevator e to be present at floor f .
 - *disembark*(p, e, f): passenger p exits the elevator e at floor f , requiring the passenger p to be inside the elevator e and elevator e to be at the intended floor f .
 - *move*(e, f_1, f_2): elevator e moves from floor f_1 to floor f_2 , requiring the elevator e to be at floor f_1 , and floor f_2 to be reachable with elevator e .

5.2.1 Data Preprocessing

LAML takes as input a set of state transitions (s, a, s') , where s and s' represent current and next states, respectively, and a denotes the action taken. Each action is labeled with a descriptive identifier and associated grounded parameters. Each state s consists of N different grounded state variables, represented as Boolean values, each characterized by a name and a unique set of arguments composed of grounded symbols. States may also contain unknown values, denoted as empty within the dataset.

The first step involves data preprocessing of state variables, and it is formally described in Algorithm 3. For each state variable x in the variable set, a separate dataset is generated. This process involves projecting each variable to reduce the dataset's dimensionality, resulting in the creation of one-dimensional datasets, each representing individual state variables. Thus, for a system comprising N grounded state variables, N one-dimensional datasets are produced (line 9).

Each row within these datasets represents a Boolean value of the state variable and its corresponding action. To refine the dataset further, only pairs $(s[x], a)$ are retained, where $s[x]$ is the value of x in state s , such that the set of grounded parameters in the action a includes all grounded arguments of the state variable x (line 8-9). This is done by using the function

$$Params(x = (\text{name } p_1 \dots p_n)) = \{p_1, \dots, p_n\}$$

which extracts the parameters of a state variable or action x into a set. This refinement process ensures that only state variables with matching objects are kept, except for state variables with no arguments, which are always included.

Note that only the current state s is used in this process; the next state s' is ignored. This is because these datasets are used to learn preconditions, i.e., the state context when an action is applied, rather than its effects.

Algorithm 3 Data preprocessing state variables

Require: Dataset \mathcal{D} consisting of transitions (s, a, s')

- 1: Initialize $G^s \leftarrow []$
- 2: Extract all distinct state variables X from \mathcal{D}
- 3: **for** each state variable $x \in X$ **do**
- 4: $D_x \leftarrow []$
- 5: **for** each $(s, a, s') \in \mathcal{D}$ **do**
- 6: **for** each state variable $x \in X$ **do**
- 7: **if** $s[x]$ is not empty **and** $\text{Params}(x) \subseteq \text{Params}(a)$ **then**
- 8: $D_x.\text{append}(\{(s[x], a) \mid (s, a, s') \in \mathcal{D}\})$
- 9: Append D_x to G^s

return G^s

Similarly, a dataset is generated for each grounded action, retaining only the state variables applicable to that action based on their groundings (Algorithm 4). In this case, the entire transition (s, a, s') is retained in the dataset. This enables the subsequent extraction of effects by comparing the state before and after the action. In detail, for every grounded action, all state variables x from the current state s such that $\text{Params}(x) \subseteq \text{Params}(y)$, are collected thus focusing on the context in which y is applied, checking whether all the parameters in the grounded state variable x are also parameters in the grounded action y .

Algorithm 4 Data preprocessing actions

Require: Dataset \mathcal{D} consisting of transitions (s, a, s')

- 1: Initialize $G^a \leftarrow []$
- 2: Extract all distinct grounded actions Y from \mathcal{D}
- 3: **for** each action $y \in Y$ **do**
- 4: $D_y \leftarrow []$
- 5: **for** each grounded action $y \in Y$ **do**
- 6: **for** each $(s, y, s') \in \mathcal{D}$ **do**
- 7: **if** $s[x]$ is not empty **and** $\text{Params}(x) \subseteq \text{Params}(y)$ **then**
- 8: $D_y.\text{append}(\{(s[x], y, s'[x]) \mid (s, y, s') \in \mathcal{D}\})$
- 9: Append D_y to G^a

return G^a

Example Considering the elevator scenario with 1 elevator, 3 passengers, and 4 floors, we generate 23 datasets: 12 for grounding the condition *passengerAt*,

3 for grounding *passengerIn*, 4 for grounding *elevatorAt*, and 4 for grounding *reachable*. Each dataset contains grounded actions corresponding to the respective grounded conditions. For instance, the *passengerIn* $p_0 e_0$ dataset (see Figure 5.3) will exclusively map grounded actions containing both p_0 and e_0 , alongside their corresponding Boolean values for the *passengerIn* variable.

passengerIn $p_0 e_0$		reachable $f_0 e_0$	
s	action	s	action
F	board $p_0 e_0 f_0$	F	board $p_0 e_0 f_0$
F	board $p_0 e_0 f_1$	T	board $p_1 e_0 f_0$
F	board $p_0 e_0 f_2$	T	board $p_2 e_0 f_0$
F	board $p_0 e_0 f_3$	F	disembark $p_0 e_0 f_0$
T	disembark $p_0 e_0 f_0$	F	disembark $p_1 e_0 f_0$
T	disembark $p_0 e_0 f_1$	T	disembark $p_2 e_0 f_0$
T	disembark $p_0 e_0 f_2$	F	move $e_0 f_0 f_1$
T	disembark $p_0 e_0 f_3$	T	move $e_0 f_0 f_2$
		F	move $e_0 f_0 f_3$
		T	move $e_0 f_1 f_0$
		T	move $e_0 f_2 f_0$
		T	move $e_0 f_3 f_0$

Figure 5.3: Illustrative examples of the generated datasets for the grounded variables *passengerIn* $p_1 e_0$ and *reachable* $f_0 e_0$, within the elevator scenario with 1 elevator, 3 passengers, and 4 floors. The figure shows the Boolean values (T for True, F for False) of the grounded state variable when the corresponding action is applied.

5.2.2 Prelifting

To generalize the datasets, grounded parameters are replaced with generic placeholders, following a similar approach to PlanMiner [152]. The full procedure is outlined in Algorithm 5. Specifically, in the grounded state variables' datasets, every parameter is substituted with a generic token *arg*, while parameters in actions not appearing in state variables are replaced with a generic constant *const* (lines 6-12).

Then, state variables with identical lifted signatures are merged into a single dataset (line 13), thus creating a dataset for each generic state variable. Therefore, for a system with M state variables, the output consists of M lifted datasets.

Similarly, actions are lifted by replacing their arguments with generic tokens, and each resulting lifted action dataset L_a is added to the list L_A .

Algorithm 5 Prelifting state variables

Require: List G^s containing grounded state variable datasets D_x

- 1: Initialize $L_S \leftarrow \square$
 - 2: **for** each dataset $D_x \in G^s$ **do**
 - 3: $I \leftarrow \text{list}(\text{Params}(x))$
 - 4: $C \leftarrow \square$
 - 5: **for** each $(s, a) \in D_x$ **do**
 - 6: **for** each parameter p in a **do**
 - 7: **if** $p \in I$ **then**
 - 8: Replace p with arg_i , where i is the position of p in I
 - 9: **else**
 - 10: **if** $p \notin C$ **then**
 - 11: $C.\text{append}(p)$
 - 12: Replace p with const_i , where i is the position of p in C
 - 13: Merge the datasets in D_x into a new dataset if x has identical lifted signatures into a new dataset L_s
 - 14: Append L_s to L_S
 - 15: **return** List L_S containing M lifted datasets L_s
-

Example In the elevator scenario, the state variable $\text{elevatorAt}(e_0, f_1)$, where e_0 and f_1 represent a specific elevator and a specific floor respectively, is generalized to $\text{elevatorAt}(\text{arg}_1, \text{arg}_2)$, replacing e_0 and f_1 with generic argument tokens (see Figure 5.4). Actions involving this variable are updated accordingly. For instance, $\text{board}(p_2, (e_0, f_1))$ is generalized to $\text{board}(\text{const}, \text{arg}_1, \text{arg}_2)$, where $\text{arg}_1, \text{arg}_2$ represent generalized placeholders for the elevator and floor and const represents a generic entity (the passenger). Similarly, the state variable $\text{elevatorAt}(e_0, f_2)$ is also transformed into $\text{elevatorAt}(\text{arg}_1, \text{arg}_2)$, and the associated action $\text{board}(p_1, e_0, f_2)$ is converted to $\text{board}(\text{const}, \text{arg}_1, \text{arg}_2)$. Despite referring to different floors and passengers in their grounded forms, these actions are combined into a single generalized dataset describing the predicate elevatorAt .

The same abstraction approach applies to the actions dataset. For example, the action $\text{move}(e_0, f_1, f_2)$ becomes $\text{move}(\text{arg}_1, \text{arg}_2, \text{arg}_3)$. Correspondingly, state variables within this dataset are updated as well; for instance, $\text{reachable}(f_1, e_0)$ transforms into $\text{reachable}(\text{arg}_2, \text{arg}_1)$. This procedure ensures that state variables and actions are consistently lifted into a generalized representation.

5.2.3 Learn Lifted Preconditions

Once the datasets have been prelifted, the C5.0 decision tree algorithm is used to learn classification rules for each lifted state variable (Algorithm 6). Unlike

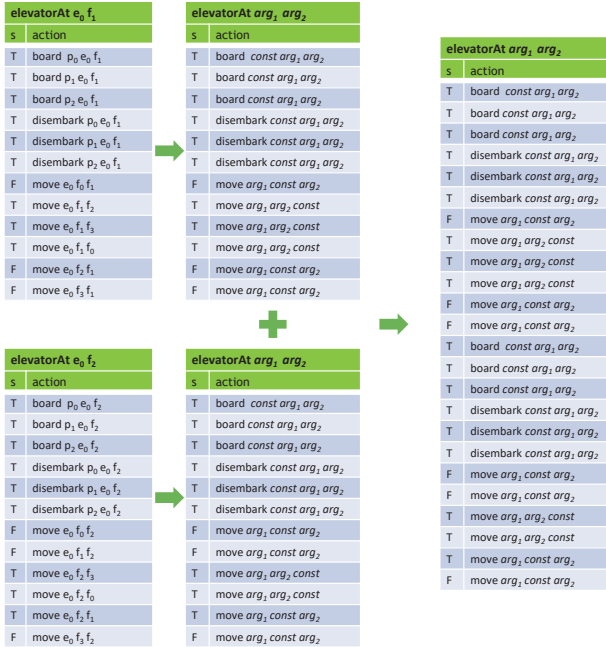


Figure 5.4: Illustrative examples of the lifting process in the elevator domain. The left side shows grounded state-action transitions, where specific elevators and floors are explicitly referenced (e.g., e_0 , f_1). The right side displays the generalized representation, where specific entities are replaced with generic argument tokens (arg_1 , arg_2). This transformation allows for a unified dataset that abstracts similar actions across different instances.

simpler methods such as rule-based heuristics or naive classifiers, decision trees can capture complex dependencies between state variables while remaining computationally efficient. Additionally, C5.0 offers robustness to noise, making it particularly effective given the noisy, real-world nature of our input data. The input to this learning phase is the list $L_S = [L_{s_1}, L_{s_2}, \dots, L_{s_M}]$ of lifted state variable datasets generated in the prelifting stage. Each L_s corresponds to a specific state variable $s \in M$ and contains tuples of the form $(s[x], a)$, where a is the lifted action taken, represented by its name and list of lifted parameters (e.g., $board(p, e, f)$ and $s[x] \in \{true, false\}$ is the Boolean value of the state variable s in the state where a is applied.

However, a limitation of C5.0 is its inability to learn from static state variables or those that remain unchanged across the entire dataset. To address this, an additional step is introduced in the data preprocessing to ensure

that decision tree learning captures meaningful patterns. Specifically, a small amount of synthetic noise is injected by randomly flipping a single value in any constant grounded variable. This prevents the variable from being entirely uninformative, allowing the model to consider it during rule induction.

More in detail, for a system with M state variables, the learning process involves training M decision trees \mathcal{T}_s using the C5.0 algorithm. Each resulting tree consists of a root node and one additional level of child nodes (i.e., 1-level decision trees) (line 2): the root node represents the state variable s , the edges denote the possible actions and the leaf nodes correspond to the predicted value of the state variable x when the actions on the edge are applied (see Figure 5.5). This process identifies which actions require specific state variables to hold for their execution. Since the state variables are binary, the learning problem is framed as identifying in which actions each specific condition is *true*.

To convert the decision tree into a set of grounded preconditions, the following steps are applied:

- Each decision tree is traversed from the root to every leaf that corresponds to the *true* class, indicating the actions where the condition holds (lines 5-7).
- For each action, a conjunction of all true state variables associated with it is formed (lines 8-9).

Through this method, all the conditions that must be met to perform an action are systematically identified.

Algorithm 6 Learn lifted preconditions

Require: List L_S of lifted state variable datasets L_s

- 1: Initialize an empty set \mathcal{T} to store decision trees
 - 2: $\mathcal{T} \leftarrow \{C5.0(L_s) | L_s \in L_S\}$ //Train a set of decision trees
 - 3: $\forall a \in A : \mathcal{P}_a \leftarrow \emptyset$
 - 4: **for** $\mathcal{T}_s \in \mathcal{T}$ **do**
 - 5: **for** each leaf node i in \mathcal{T}_s **do**
 - 6: **if** *true* class **then**
 - 7: Extract the set of actions A_s leading to i
 - 8: **for** each action $a \in A_s$ **do**
 - 9: $\mathcal{P}_a \leftarrow \mathcal{P}_a \cup \{s\}$ //add s as a precondition of a
 - return** $\{\mathcal{P}_a | a \in A\}$
-

Example In the elevator scenario, four sets of rules are derived, corresponding to the four state variables. For the state variable $passengerAt(arg_1, arg_2)$, the relevant lifted action where it holds true is $board(arg_1, const, arg_2)$. Similarly, for the state variable $elevatorAt(arg_1, arg_2)$, the lifted actions where

it holds true include $move(arg_1, arg_2, const)$, $board(const, arg_1, arg_2)$, and $disembark(const, arg_1, arg_2)$.

To determine the preconditions of an action, all conditions where the corresponding state variable holds true are considered. The generic action $board(arg_1, const, arg_2)$ appears in the rule set of $passengerAt(arg_1, arg_2)$, while $board(const, arg_1, arg_2)$ appears in the rule set of $elevatorAt(arg_1, arg_2)$. By merging these rules and mapping their arguments, the lifted preconditions of the action $board(arg_1, arg_2, arg_3)$ are identified as: $passengerAt(arg_1, arg_3)$ and $elevatorAt(arg_2, arg_3)$

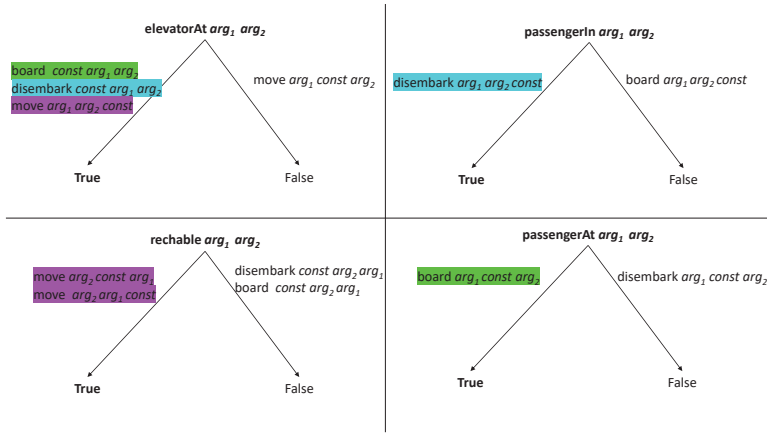


Figure 5.5: Decision trees used for precondition learning in the elevator domain. Each tree represents one state variable ($elevatorAt$, $passengerIn$, $reachable$, $passengerAt$) and identifies the lifted actions where the variable holds *true* (left branch) or *false* (right branch). The highlighted nodes correspond to actions influenced by each state variable, and merging these rules leads to the generalized lifted preconditions of actions.

5.2.4 Learn Lifted Effects

State variable changes caused by actions are determined using frequency analysis, as outlined in Algorithm 7. The input to this process is the set of lifted action datasets L_A . For each action, and for each state variable within its context, the value of the variable is tracked to determine whether it remains consistently true or false, or if it changes during action execution and the occurrences of each case are counted (lines 4-13).

Given the noise in the data, the effect is determined using an *argmax* function to identify the most frequent change pattern (line 15):

- If the most frequent pattern is no change, the variable is not considered an effect of the action.
- If the most frequent change is to *true*, the variable is marked as a positive (add) effect (lines 16-17).
- If the most frequent change is to *false*, the variable is marked as a negative (delete) effect (lines 18-19).

This statistical approach accounts for noise and mitigates uncertainty.

Algorithm 7 Learn lifted effects

Require: List L_A of L_a lifted action datasets

```

1:  $\mathcal{E} \leftarrow \{\forall a \in A, E_a \leftarrow \emptyset\}$ 
2: for each  $L_a$  in  $L_A$  do //for each action a
3:    $\mathcal{F}[x, c] \leftarrow 0 \forall x \in X, c \in \{const+, const-, eff+, eff-\}$ 
4:   for each transition  $(s, a, s') \in L_a$  do
5:     for each state variable  $x$  in  $s$  do
6:       if  $s[x] = s'[x]$  and  $s[x] = true$  then
7:          $\mathcal{F}[x, const+]_+ = 1$ 
8:       else if  $s[x] = s'[x]$  and  $s'[x] = false$  then
9:          $\mathcal{F}[x, const-]_+ = 1$ 
10:      else if  $s[x] = false$  and  $s'[x] = true$  then
11:         $\mathcal{F}[x, eff+]_+ = 1$ 
12:      else if  $s[x] = true$  and  $s'[x] = false$  then
13:         $\mathcal{F}[x, eff-]_+ = 1$ 
14:      for each  $x \in \mathcal{F}$  do
15:         $f \leftarrow \arg \max_c \mathcal{F}[x, c]$ 
16:        if  $f = eff+$  then
17:           $E_a \leftarrow E_a \cup \{x^+\}$  //add x as an positive effect of a
18:        else if  $f = eff-$  then
19:           $E_a \leftarrow E_a \cup \{x^-\}$  //add x as an negative effect of a
20:       $\mathcal{E} \leftarrow E_a$ 
  
```

return \mathcal{E}

Example: In the elevator running example, consider the action $board(arg_1, arg_2, arg_3)$. When this action is executed, the state variable $passengerAt(arg_1, arg_3)$ typically transitions from *true* (before the action) to *false* (after the action). On the other hand, $passengerIn(arg_1, arg_2)$ changes from *false* to *true*. These transitions happen consistently when the data are noise-free. However, due to noisy data, there might be instances where these state variables do not change as expected. To handle this, we count occurrences of each possible state transition and select the most frequent one, ensuring robustness against noise.

5.2.5 Generate PDDL Domain

The learned action models are translated into PDDL format, assuming prior knowledge of the action signature, that is, the names of actions and their argument types, but not their preconditions or effects. This assumption is necessary to correctly structure actions within the PDDL domain. In practice, it remains a lightweight assumption, as action names and argument types are typically available from domain specifications, execution logs, or system documentation.

Unlike many existing methods that require a full domain skeleton, which is a predefined domain containing object types, predicates, and action schemas with empty preconditions and effects, this approach does not depend on additional structured knowledge beyond the observed actions themselves.

This translation enables seamless integration with AI planners, allowing the learned action models to be directly utilized for automated planning tasks. The encoding follows PDDL2.0, though other representation languages could be adopted with minimal modifications.

The final representation of the learned action models is illustrated in Figure 5.6, demonstrating how preconditions and effects are structured within the domain.

```
(define (domain elevator)
  (:requirements :strips)
  (:types e f p)
  (:predicates
    (elevatorAt ?e - e ?f - f)
    (reachable ?f - f ?e - e)
    (passengerAt ?p - p ?f - f)
    (passengerIn ?p - p ?e - e)
  )
  (:action move
    :parameters (?e - e ?f0 - f ?f1 - f)
    :precondition (and (elevatorAt ?e ?f0) (reachable ?f1 ?e))
    :effect (and (not (elevatorAt ?e ?f0)) (elevatorAt ?e ?f1))
  )
  (:action board
    :parameters (?p - p ?e - e ?f - f)
    :precondition (and (elevatorAt ?e ?f) (passengerAt ?p ?f))
    :effect (and (not (passengerAt ?p ?f)) (passengerIn ?p ?e))
  )
  (:action disembark
    :parameters (?p - p ?e - e ?f - f)
    :precondition (and (elevatorAt ?e ?f) (passengerIn ?p ?e))
    :effect (and (not (passengerIn ?p ?e)) (passengerAt ?p ?f))
  )
)
```

Figure 5.6: Example of the LAML learned PDDL domain in the elevator domain

5.3 Comparison with NOLAM

While both LAML and NOLAM aim to learn planning domain models from noisy execution traces, their fundamental assumptions and learning mechanisms are quite different.

NOLAM is based on a Bayesian probabilistic framework. It models the uncertainty in observations by treating the truth value of each possible precondition and effect as a random variable. A graphical model is constructed over these variables, and learning is achieved by computing the posterior probability of each condition being part of an operator’s definition. This allows NOLAM to reason explicitly about uncertainty and noise in the input traces.

However, to perform this inference, NOLAM requires the noise level to be known and specified as a parameter. The method assumes that sensor noise can be modeled as a Bernoulli process with a fixed probability e , and this probability must be provided by the user. The model uses this parameter to reason about the likelihood of observed state transitions being flipped or corrupted. Incorrect assumptions about this noise parameter can reduce the accuracy of the learned model.

In contrast, LAML follows a frequentist approach that does not require any prior knowledge of the noise level. Instead of modeling uncertainty probabilistically, LAML learns from consistent patterns in the data. Preconditions are extracted using decision tree learning, and effects are inferred through frequency analysis of observed state changes. This makes LAML more flexible in situations where the characteristics of the noise are unknown or variable across the dataset.

This subsection provides the conceptual backdrop for the more detailed evaluation in Chapter 6.

5.4 Summary and Outlook

This chapter introduced LAML, an approach for learning lifted planning domains from execution traces in noisy environments. The method operates by analyzing sequences of state transitions and identifying preconditions and effects that explain observed action behavior. The resulting action models are intended to support automated planning and enable further analysis of system behavior.

The next chapter addresses this challenge by examining the evaluation of learned planning domains, a task that lacks standardized metrics and methodologies. It begins with a review of the key difficulties in assessing learned planning domain quality and then introduces the multi-faceted evaluation strategy applied in this thesis. This provides a structured basis for analyzing the strengths and limitations of LAML and contributes to a broader understanding of how to assess learned planning models in practice.

Chapter 6

Evaluating Learned Planning Domains

The evaluation of learned models is a fundamental yet challenging aspect of knowledge engineering. While Chapter 4 addressed the evaluation of Behavior Trees (BTs), the focus here shifts to assessing the learning algorithm responsible for learning planning domains. In this context, the planning domain is not the ultimate target of evaluation, but rather the output artefact through which the quality and generalizability of the learning process is inferred.

This distinction is crucial: whereas the BT evaluation was focused on how well a synthesised BT fulfilled its task-oriented role, the present evaluation seeks to answer a different question: *how well does the learning algorithm reconstruct a planning domain model that generalises correctly and supports effective planning?* Thus, the quality of the learned domain serves as a proxy for evaluating the efficacy and reliability of the underlying learning mechanism.

Evaluating planning domains remains a complex problem due to the absence of universally accepted metrics. Different evaluation approaches offer distinct insights, yet each comes with inherent limitations, making it difficult to establish a single, definitive metric. A comprehensive assessment of planning domains extends beyond structural correctness to encompass their functional performance, particularly their ability to generate valid and effective plans. A learned domain must accurately represent the planning environment while supporting the derivation of valid plans.

This chapter explores the key challenges in evaluating learned planning domains and reviews the state of the art on the difficulties associated with their assessment. It then introduces the evaluation methods applied in this thesis, each targeting a distinct aspect of domain quality. These include comparison with reference domains, plan generation feasibility, comparison with historical plans, simulation-based evaluation, plan validation, and human expert

evaluation. Each method is analysed in terms of its applicability, strengths, and limitations, providing a structured perspective on how to evaluate learned planning models in practice.

6.1 Related Work

Evaluating the quality of planning domain models has been approached from multiple perspectives, ranging from structural comparison to semantic validation and human interpretability. Each of these approaches contributes to a more comprehensive understanding of how learned models align with real-world applications and theoretical expectations. In this thesis, however, the primary goal is not to evaluate the learned domain as an artefact in itself, but to use it as a means to assess the performance of the learning algorithm that produced it. The planning domain is treated as the output of a learning process, and its quality is interpreted as indicative of how well the underlying algorithm has captured the structure and dynamics of the environment.

A foundational line of research focuses on defining general criteria for assessing the quality of planning knowledge models. McCluskey et al. [112] formalise semantic quality in terms of correctness, completeness, and adequacy, emphasising the need for models that not only capture the domain structure but also function effectively within a planner. Vallati & McCluskey [167] extend this framework by distinguishing syntactic, operational, and pragmatic quality. These frameworks provide formal guidance for evaluating planning domain models.

To improve domain quality, other methods focus on refining domain models based on execution data. Machine-learning-based approaches detect discrepancies between expected and observed outcomes in execution traces and revise models accordingly [102]. This aligns with historical plan comparison, where learned models are validated against real-world execution data. These techniques inform evaluation strategies that assess a model's ability to generate plans that match historical records. From the perspective adopted in this thesis, such evaluations contribute directly to diagnosing the accuracy and robustness of the learning process.

Understanding how humans interpret planning domain models is another important aspect of evaluation. Studies on hierarchical planning domains investigate the role of preconditions and effects in aiding user comprehension, demonstrating that providing inferred conditions for compound tasks improves comprehension of learned models [123]. Similarly, Poels et al. [133] develop a Perceived Semantic Quality (PSQ) measure for evaluating how well a model conveys real-world semantics. These insights suggest that expert evaluation should consider not only structural and functional correctness but also model interpretability. In the present work, such human-centered evaluations are used not only to assess usability but also as an indirect indicator of the quality of

the learning algorithm itself, based on how intuitive and coherent the resulting domain appears to domain experts.

A central challenge in this thesis is evaluating how learned models compare to existing ones. Several approaches focus on structural comparison to measure similarity between domain models. Methods based on Answer Set Programming (ASP) [21] formalise domain model similarity as a graph edit distance problem, enabling the identification of structural deviations. Shoenen & McCluskey [154] define weak and strong equivalence to formally assess domain model similarity. Strong equivalence ensures logical identity up to naming, requiring that every corresponding grounded instance of two strongly equivalent lifted domain models also exhibits strong equivalence. However, as generating all possible grounded problems is infeasible, they define weak equivalence, which considers two models functionally identical if they produce the same valid plans. These methods establish a formal foundation for evaluating learned models in relation to reference domains, aligning with structural correctness-based evaluation metrics.

6.2 Evaluation Methods

Building upon the challenges outlined in the previous sections, this work investigates a range of evaluation methods designed to assess different aspects of learned planning domain models. Each evaluation method provides insights into different aspects of a learned planning domain, and its applicability depends on the available resources and conditions, such as the presence of a simulator or past execution data.

The evaluation methods considered include:

- **Comparison with reference domain**

This method evaluates how well the learned domain aligns with an existing reference model, such as International Planning Competition (IPC) benchmark domains. The structural accuracy is assessed using precision and recall, measuring the extent to which the learned model correctly captures preconditions and effects present in the reference domain. However, reference domains are often unavailable in real-world applications, limiting the generalizability of this approach.

- **Plan generation feasibility** This approach assesses whether the learned model can generate plans for given problems. This can be assessed by computing the ratio of problems for which a solution can be generated using the learned model. A high ratio suggests broad applicability, though it does not guarantee the correctness or executability of the generated plans.

- **Comparison with historical plans** By comparing generated plans with previously executed real-world plans, this method evaluates how closely a learned domain aligns with historical decision-making. Distance between the two plans, such as graph edit distance or other plan similarity metrics, can quantify deviations between generated and historical plans. However, historical plans may not always be optimal, and multiple valid solutions may exist for the same problem.
- **Simulation-Based Evaluation** In this approach, the learned domain is tested by executing generated plans in a simulated environment, assessing whether they achieve their intended goals. The success ratio of plans reaching their goal states is used as a key metric. Beyond binary success rates, simulation environments can provide domain-specific metrics that offer deeper insights into model quality. For example, in robotic domains, these may include energy consumption, time to goal, number of collisions, or safety constraint violations. In logistics settings, simulation may track resource utilization, delivery time, or throughput. By tailoring evaluation metrics to domain requirements, simulation-based assessments can more accurately reflect operational performance and trade-offs. However, the accuracy of this evaluation depends on the fidelity of the simulator, which may not perfectly reflect real-world conditions.
- **Plan validation** Plan validation ensures that a generated plan is valid when executed under the constraints of the domain. This is measured by computing the ratio of valid plans generated by the learned model that remain executable in the reference domain. This method relies on plan validation tools, but is limited by the availability of reference domains for comparison.
- **Human expert evaluation** Human experts manually assess the learned domain and its generated plans for correctness and relevance. While this method accounts for real-world nuances and domain-specific expertise, it is time-consuming, subjective, and lacks scalability. Expert evaluation is most useful in scenarios where automated validation methods are insufficient.
- **Real-World Execution Evaluation** In scenarios where system deployment is feasible, plans generated from the learned domain can be executed directly in the real world, with outcomes monitored to assess correctness and effectiveness. This form of evaluation shares similarities with reinforcement learning, where agents iteratively improve through real-world interactions. Metrics may include task completion rate, execution time, error frequency, or human intervention needs. However, this method is associated with higher cost, risk, and limited scalability, particularly during early model development or in safety-critical domains.

Despite the availability of various evaluation techniques, several challenges continue to hinder the effective assessment of learned planning domains. Earlier methods [4, 117] relied on reference domain evaluation, but more recent approaches [116, 152] have integrated plan validation to assess functional correctness. However, reference-based methods remain limited by the availability of predefined domains. NOLAM [99] further incorporates plan generation feasibility, allowing evaluation without a reference domain by testing the model’s ability to generate plans for new problems. Table 6.1 provides a structured comparison of the evaluation methods discussed, outlining their key characteristics, associated metrics, advantages, and limitations. This serves as a valuable reference for selecting the most suitable evaluation approach based on the specific requirements and constraints of a given planning domain.

6.3 Evaluation Metrics

The evaluation of learned planning domains requires well-defined metrics that quantitatively assess different aspects of model quality. This work employs a selection of metrics that address both structural accuracy and practical applicability, ensuring alignment with the evaluation methods discussed in the previous sections. In practice, domain validation is performed over a set of problems, not just a single instance. Consequently, some computed metrics are normalized to provide a more representative assessment of model performance.

Comparison with Reference Domain

To determine how well a learned domain structure aligns with an established reference model, precision and recall are employed as key metrics. Precision measures how accurately the learned conditions match the ground truth, while recall assesses how comprehensively the learned domain captures the benchmark domain. Precision and recall can be computed for preconditions and effects. Additionally, they can be defined separately for positive preconditions (Pre+), negative preconditions (Pre-), positive effects (Eff+), and negative effects (Eff-), as well as for the entire action model, considering all preconditions and effects together. Mathematically, precision is defined as:

$$p = \frac{c}{(c + l)},$$

where c is the number of conditions in both the learned and the benchmark domains, and l is the number of conditions in the learned but not in the benchmark domain. On the other hand, recall is defined as:

$$r = \frac{c}{(c + b)},$$

where b represents the number of conditions in the benchmark domain but not in the learned domain.

These definitions ensure that evaluation accounts for both the accuracy of learned conditions and the coverage of reference conditions, offering a measure of structural alignment.

Plan Generation Feasibility

Plan generation feasibility evaluates whether the learned domain can successfully generate valid plans for a given set of planning problems. This metric assesses the model’s ability to produce solutions across different problem instances, providing insight into its general applicability. The feasibility of plan generation is measured using the ratio of solvable problems, defined as:

$$SR_p = \frac{\sum_{i=1}^N \mathbf{1}(P_{g_i} \neq \emptyset)}{N}, \quad (6.1)$$

where N represents the total number of evaluated planning problems, P_{g_i} is the generated plan for the i^{th} problem, and $\mathbf{1}$ is an indicator function that returns 1 if a plan was successfully generated and 0 otherwise.

A higher ratio suggests that the learned model is capable of generating solutions for a larger set of problems. However, this measure does not assess the correctness, optimality, or executability of the generated plans, necessitating further validation.

Comparison with Historical Plans

Evaluating a learned model’s consistency with real-world decision-making requires comparing its generated plans against historical execution data. The similarity between these plans is measured using plan distance metrics in order to quantify how much a generated plan deviates from a historical plan.

The similarity between these plans can be assessed using two complementary metrics:

- A binary metric, which assesses whether a generated plan exactly matches the corresponding historical plan, computed as:

$$SR_m^b = \sum_{i=1}^N \frac{\mathbf{1}(P_{g_i} \neq \emptyset) \cdot \mathbf{1}(P_{g_i} = P_{h_i})}{\mathbf{1}(P_{g_i} \neq \emptyset)} \quad (6.2)$$

where N is number problems, P_{g_i} and P_{h_i} are the generated and historical plans for the i^{th} problem respectively, $\mathbf{1}$ is an indicator function that returns 1 if a plan was successfully generated and 0 otherwise and $\mathbf{1}(P_{g_i} = P_{h_i})$ is an indicator function that returns 1 if the plans are identical and 0 otherwise.

An $SR_m^b = 1$ signifies that all generated plans are identical to historical executions, whereas a value near 0 indicates substantial deviation. Therefore, SR_m^b measures how often exact matches occur.

- A quantitative distance metric, which captures how much a generated plan deviates from the corresponding historical plan. The distance function $D(P_{g_i}, P_{h_i})$ can be computed using various methods:
 - Edit Distance: The number of insertions, deletions, or substitutions required to transform one plan into another.
 - Action Overlap Ratio: The fraction of shared actions between the generated and historical plans.
 - Execution Cost Difference: The difference in cumulative execution cost between the two plans.

The normalized plan distance metric is computed as:

$$SR_m^q = \sum_{i=1}^N \frac{\mathbf{1}(P_{g_i} \neq \emptyset) \cdot D(P_{g_i}, P_{h_i})}{\mathbf{1}(P_{g_i} \neq \emptyset)} \quad (6.3)$$

where $D(P_{g_i}, P_{h_i})$ represents the distance between the generated and historical plans, normalized to a range of $[0, 1]$, with 0 indicating an exact match and 1 representing maximum divergence.

A lower value of SR_m^q signifies stronger alignment with historical plans, whereas a higher value indicates greater deviations. Therefore, SR_m^q quantifies deviations when exact matches do not occur.

Plan Validation

Plan validation assesses whether the plans generated using the learned domain are valid. A plan is considered valid if executing its actions sequentially from the initial state leads to a final state that matches the objective state. The validation ratio is defined as:

$$SR_v = \sum_{i=1}^N \frac{\mathbf{1}(P_{g_i} \neq \emptyset) \cdot \text{Valid}(P_{g_i})}{\mathbf{1}(P_{g_i} \neq \emptyset)} \quad (6.4)$$

where N is the number of problems, $\mathbf{1}$ is an indicator function that returns 1 if a plan was successfully generated and 0 otherwise, and $\text{Valid}(P_{g_i})$ is a binary function that returns 1 if the generated plan is valid and 0 otherwise.

While this metric ensures the validity of generated plans, its effectiveness depends on the availability of reference domains.

Table 6.1: Comparison of advantages (+) and limitations (-) of evaluation methods.

Method	Description	Metric	Remarks
Comparison with Reference Domain	Assesses the similarity between the learned domain and a predefined reference domain.	Precision and Recall	(+) Enables objective comparison; supports reproducibility; facilitates benchmarking (-) Does not ensure functional validity; reference domains may be unavailable; may not guarantee plan correctness; may miss real-world complexity
Plan Generation Feasibility	Tests whether the learned domain can generate a valid plan for a given problem and goal.	Ratio of solvable problems	(+) Verifies basic functionality; identifies missing elements; computationally lightweight (-) Does not assess plan correctness or quality; cannot detect flawed constraints
Comparison with Historical Plans	Evaluates generated plans by comparing them to previously executed real-world plans.	Edit distance, deviation	(+) Uses real-world data; compares new plans to past decisions; measures planning consistency (-) Historical plans may not be optimal; multiple valid plans may exist; errors in historical data can mislead evaluation; limited ability to generalise to new scenarios
Simulation-Based Evaluation	Evaluates whether generated plans successfully achieve the goal in a simulated environment.	Ratio of successful goal completions in simulation	(+) Assesses goal achievement; enables large-scale testing; supports tailored evaluation (-) Depends on simulator fidelity; may not reflect real-world execution; may miss real-world unpredictability
Plan Validation	Checks if a generated plan remains valid when compared to a reference domain, without actual execution.	Ratio of valid solutions relative to the reference domain	(+) Ensures plan validity, consistency, and executability; scalable automation (-) Requires a reference model; does not assess efficiency or optimality; cannot guarantee completeness or absence of biases
Human Expert Evaluation	Experts manually assess the correctness of the learned domain and generated plans.	Qualitative expert judgment	(+) Accounts for real-world complexities; identifies errors overlooked by automated methods; works when formal references or simulations are unavailable (-) Time-consuming and costly; subjective; not scalable or reproducible
Real-World Execution Evaluation	Executes generated plans in the physical world to assess actual outcomes.	Task success rate; runtime metrics	(+) Captures real-world complexity and unforeseen interactions; supports iterative refinement (-) Resource-intensive; may be risky or unsafe; difficult to repeat and scale

6.4 LAML Evaluation

The implementation of LAML was developed using Python 3.8 for the core functionality, with the C5.0 decision tree algorithm provided via the C5.0 package¹ in R.

The evaluation is designed to assess both the structural accuracy of the action models learned by LAML under varying levels of noise in the input traces and their practical effectiveness in solving planning problems. Accuracy is measured against a reference model using standard structural metrics, namely precision and recall. In addition, planning performance is evaluated through metrics that capture plan generation success, similarity to historical plans, and validation success rates.

The results are compared against NOLAM [99] using the exact same execution traces² to learn planning domains to ensure consistency in the experimental setup. These traces are derived from 23 classical planning domains featured in past International Planning Competitions (IPCs), as summarized in Table 6.2.

Trace generation follows the method described by Lamanna & Serafini [99]. For each planning problem, a solution plan was computed using the ground-truth action model provided by the IPC domain. Planning was performed using FastDownward [69], configured with a lazy greedy best-first search strategy guided by the context-enhanced additive heuristic [43] and the FastForward heuristic [71]. Each computed plan was then executed from the initial state to obtain a set T of 10 execution traces per domain. The generated traces span a wide range of complexity, containing between 1 and 70 transitions, 3 to 58 objects, and 8 to 687 ground atoms. To simulate varying levels of noise, the traces were further modified by randomly flipping the truth value of each ground state variable with probability $p \in \{0, 0.1, 0.2, 0.3, 0.4\}$. While the domain learning is conducted using plan traces, this is not a requirement of the method. LAML is designed to operate on any form of state-action-state sequences, including those produced by random action walks or real-world execution logs. The method does not assume access to goals, plan optimality, or known domain structure. What is required is that transitions between states are observed, so that action effects and preconditions can be inferred. In this sense, LAML is agnostic to how the data is generated and can be applied in both synthetic and realistic data collection settings. To evaluate planning performance (plan generation success, comparison with historical plans, and plan validation), we used the same set of 10 test problems per domain that were randomly generated in the NOLAM study³. These problems are distinct from those used to produce the training traces, ensuring that the evaluation reflects

¹<https://CRAN.R-project.org/package=C50>

²<https://github.com/LamannaLeonardo/NOLAM/tree/main/Analysis/Input%20traces/NOLAM>

³Available as additional material at <https://openreview.net/forum?id=nSG14pICyD>

generalization beyond the data seen during learning. Each test problem was solved using the learned action models, with a CPU time limit of 60 seconds per problem. This time limit was sufficient to solve all problems using the ground-truth action model.

The results reported for NOLAM were successfully replicated and used as a baseline for comparison. This choice provides an indirect benchmark against other approaches, such as PlanMiner [152], ALICE [117], and a frequentist baseline, all of which were evaluated in the original NOLAM publication. As NOLAM has been shown to consistently outperform these approaches, additional comparisons were deemed unnecessary. An intuitive summary of the conceptual differences between LAML and NOLAM is presented in Section 5.3.

Domain	#objects	$ \mathcal{S} $	$ \mathcal{A} $	max arity \mathcal{S}	max arity \mathcal{A}
driverlog	6	6	6	2	4
n-puzzle	2	1	3	1	2
transport	7	5	3	2	3
tpp	8	7	3	3	7
hanoi	3	3	1	2	3
gripper	4	4	3	2	3
elevators	6	6	5	2	4
floortile	4	10	7	2	3
zenotravel	6	4	5	2	4
depots	10	6	5	2	4
ferry	2	3	3	1	2
satellite	6	10	6	3	3
spanner	6	6	3	2	3
gold-miner	1	2	7	2	2
nomystery	6	6	3	3	2
blocksworld	1	5	4	2	3
barman	10	15	12	2	4
parking	2	5	4	2	2
rover	7	25	9	3	5
matching-bw	2	10	10	2	2
sokoban	3	5	2	2	2
grid	3	9	5	2	4
miconic	6	4	2	2	2

Table 6.2: Details of the IPC classical planning domains used for the LAML evaluation. For each domain (1st column), the table reports: the number of object types (2nd column), the number of predicate (state variable) names (3rd column), the number of action schemas (4th column), the maximum predicate arity (5th column), and the maximum action arity (6th column).

6.4.1 Comparison with Reference Domain

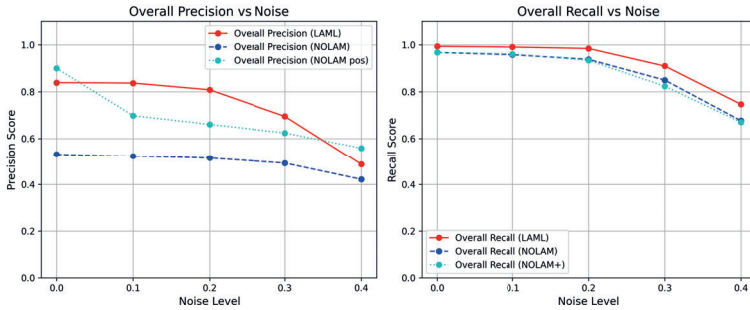


Figure 6.1: Overall precision and recall metrics for LAML and NOLAM at different noise levels. The left (right) plot illustrates the trend of precision (recall) as noise increases.

Noise	Precision Pre+		Recall Pre+		Precision Eff+		Recall Eff+		Precision Eff-		Recall Eff-		Overall Precision		Overall Recall	
	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM	LAML	NOLAM
0	0.79	0.83	1.00	0.97	0.99	0.97	1.00	0.97	0.90	0.90	0.92	0.92	0.84	0.53	0.99	0.97
0.1	0.80	0.81	0.99	0.96	0.98	0.96	1.00	0.97	0.89	0.88	0.91	0.91	0.84	0.52	0.99	0.95
0.2	0.79	0.81	0.99	0.94	0.94	0.93	0.98	0.95	0.86	0.83	0.91	0.88	0.81	0.51	0.98	0.94
0.3	0.75	0.78	0.90	0.82	0.80	0.82	0.95	0.90	0.76	0.75	0.85	0.83	0.69	0.49	0.91	0.85
0.4	0.63	0.68	0.74	0.71	0.53	0.56	0.76	0.64	0.50	0.48	0.67	0.62	0.49	0.42	0.75	0.68

Table 6.3: Precision and recall metrics for different noise levels

The performance metrics, averaged across the 23 benchmark domains from the IPC, are reported in Table 6.3, with the overall trends of precision and recall shown in Figure 6.1. The results show that LAML excels in recall for both positive and negative preconditions (Pre+ and Pre-), as well as for the precision of positive effects (Eff+). However, precision for positive preconditions (Pre+) is slightly lower compared to NOLAM, primarily due to LAML’s tendency to include some additional (not strictly necessary) positive preconditions. In comparison, NOLAM tends instead to learn many unnecessary negative preconditions that lead to a significant drop in overall precision. To further investigate this effect, LAML is also compared against a variant of NOLAM (denoted NOLAM+) in which negative preconditions are excluded, as NOLAM’s authors consider them to be often implicitly defined by positive preconditions and typically not present in IPC domains. When excluding negative preconditions, LAML continues to outperform NOLAM+ at intermediate noise levels (0.1-0.3), while NOLAM+ performs slightly better under noise-free (0.0) and high-noise (0.4) settings. However, NOLAM+ represents an artificial setting, as the removal of negative preconditions alters the learned model’s structure

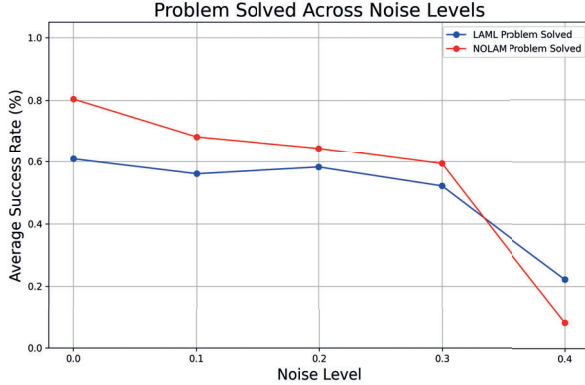


Figure 6.2: Problem resolution success rate across noise levels

in a way that may not be applicable in real-world scenarios. While this adjustment simplifies the model, it does not necessarily reflect a realistic learning process.

6.4.2 Plan Generation Feasibility

The problem resolution success rate (SR_p) reflects the proportion of planning problems successfully solved using the learned action models. Figure 6.2 shows the average success rate SR_p across different noise levels for both LAML and NOLAM. The results reflect the trade-off between precision and robustness to noise that LAML has been designed to make. In the noise-free setting, NOLAM exhibits a higher success rate (~ 0.83) compared to LAML (~ 0.61) due to the latter’s reduced precision in preconditions. This occasionally leads to over-constrained actions that prevent valid plan generation. For instance, in the Blocks domain, LAML erroneously learns that a block must be on the table before stacking, even though this is not always true. This is because, in the training data, block2 is more frequently on the table during the stack action, leading LAML to incorrectly learn this as a necessary precondition. This prevents LAML from generating valid plans, causing a drop in success rates. However, as noise increases, LAML demonstrates greater robustness. Its success rate decreases only slightly at moderate noise levels, whereas NOLAM’s success rate deteriorates more significantly. At noise level 0.4, both methods show reduced performance, though LAML maintains a higher success rate.

These findings illustrate that LAML sacrifices some accuracy in noise-free conditions to ensure stability and robustness under noisy conditions, a trade-off that is particularly beneficial in real-world applications where noise is inevitable. Alternatively, the reduced accuracy in the absence of noise may also

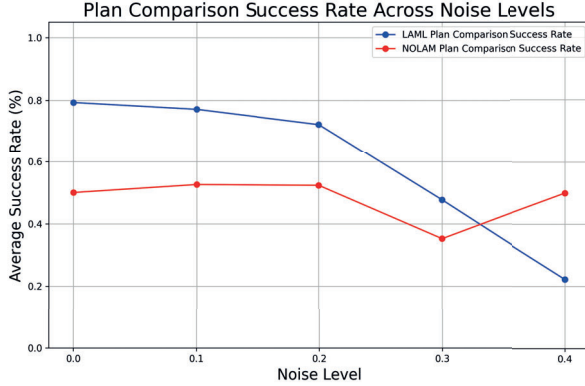


Figure 6.3: Plan comparison success rate across noise levels

reflect a need for larger or more diverse training data to fully capture the underlying domain dynamics.

6.4.3 Comparison with Historical Plans

The binary plan comparison success rate quantifies the proportion of generated plans that exactly match the corresponding ground truth plans. It is computed as the ratio between the number of successful matches and the total number of generated plans. Importantly, this metric excludes instances where the domain fails to generate any plan, focusing solely on the generated outputs. Figure 6.3 shows SR_m for both LAML and NOLAM across different noise levels, providing insight into how well the learned action models can generate plans that closely match the ground truth. The results indicate that LAML maintains a higher plan comparison success rate across increasing noise levels compared to NOLAM, up until 0.3 noise. At 0.4 noise, NOLAM’s plan comparison success rate is slightly higher, but this is due to the very few problems solved at this high noise level. Specifically, for 0.4 noise, $SR_{p,NOLAM} = 0.1$, while $SR_{p,LAML} = 0.2$. For the limited number of problems that NOLAM is able to solve, it is more efficient in terms of plan comparison success rate. LAML, on the other hand, solves more problems, but the success rate for those plans is lower. It is important to consider that in real-world scenarios, the noise level rarely reaches such extreme values as 0.4, where each sensor reading has a 40% probability of being incorrect. In the more typical conditions, where noise levels are generally lower, LAML tends to perform better than NOLAM.

Once more, these results may seem contradictory when compared with the trends in Figure 6.2, where NOLAM shows better performance at lower noise levels. As in the previous discussion, this apparent contradiction is resolved by

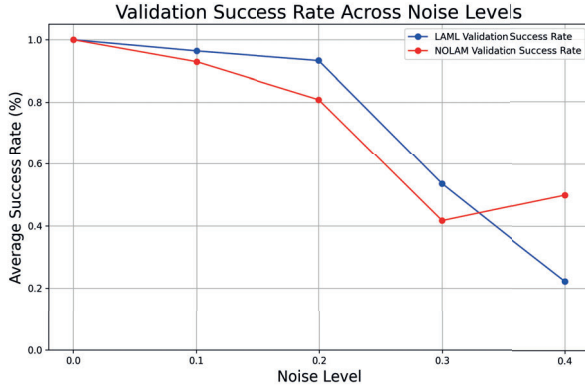


Figure 6.4: Validation success rate across noise levels

considering the differing definitions and denominators of the metrics. In particular, while the problem resolution success rate is computed over all problem instances, the binary plan comparison success rate considers only those instances where a plan is generated. As a result, SR_m focuses on the accuracy of generated plans, not on their overall coverage. Therefore, the increase in SR_m for LAML should be interpreted in the context of this restricted subset of problems.

Finally, while the plan comparison metric offers a useful approximation of model fidelity, it does not account for plan equivalence: two plans can both be correct even if they are not identical. In practical applications, comparisons to historical plans are common, as these often serve as the only available reference. Therefore, although SR_m is informative, it should be interpreted cautiously and considered alongside more holistic evaluation measures.

6.4.4 Plan Validation

The validation success rate (SR_v) represents the proportion of plans that were deemed valid by the evaluation tool VAL [73]. Figure 6.4 shows the validation success rate across different noise levels for both LAML and NOLAM. Note that the validation success rate is computed only for the cases where a plan could be successfully generated. In other words, the instances where neither method is able to produce a valid plan are considered.

In the absence of noise, both methods achieve a perfect validation success rate, indicating that when a plan is generated, it is always valid. However, as noise levels increase, the success rate for both methods declines. NOLAM experiences a deeper drop, while LAML maintains relatively higher success rates, demonstrating its superior robustness in the face of increasing noise.

This pattern continues up to 0.3 noise, where LAML still outperforms NOLAM in terms of validation success rate.

At the highest noise level (0.4), the performance of both approaches drops significantly. However, this decline should be interpreted with caution, as it is primarily driven by the very small number of problems that can be solved under such severe noise.

Note that when the noise level is 0.4 NOLAM solves only 10% and LAML 20% of the evaluation problems, respectively. In these rare cases, NOLAM shows a slightly higher validation rate per generated plan, but this is achieved over a narrower and more selective subset of problems. In contrast, LAML produces a greater number of plans overall, despite a slightly lower per-plan validation ratio.

However, such extreme noise levels, where each sensor reading has a 40% probability of being incorrect, are highly unlikely in realistic applications. In more typical scenarios with lower noise levels, LAML consistently demonstrates better validation performance, confirming its robustness and practical reliability.

At first glance, these results may appear contradictory when compared to the trends in Figure 6.2, where NOLAM shows better performance at lower noise levels. This apparent contradiction is resolved by considering the differing definitions and denominators of the metrics. In particular, while the problem resolution success rate is computed over all problem instances, the validation success rate considers only those instances where a plan is generated. As a result, SR_v focuses on the validity of generated plans, not on their overall coverage. Therefore, the increase in SR_v for LAML should be interpreted should be understood in light of the metric’s definition.

6.5 Summary and Outlook

This chapter examined the evaluation of learned planning domains, a task that presents significant challenges due to the absence of standardized metrics and the multifaceted nature of domain quality. The evaluation framework developed in this thesis incorporates both structural and functional criteria, using methods such as comparison with reference domains, plan generation feasibility, validation success rates, alignment with historical plans, simulation-based analysis, and human expert review. This comprehensive approach enables a nuanced assessment of how well learned models represent planning environments and support practical planning tasks.

Applying this evaluation framework to the LAML algorithm revealed several important findings. Across multiple metrics, LAML consistently outperformed the state-of-the-art. It achieved higher overall precision and recall, and demonstrated superior robustness to noise, a critical property for real-world deployment, where data is often incomplete or imperfect.

However, this robustness has trade-offs. LAML occasionally introduces superfluous preconditions, which can over-constrain the model and reduce its ability to generate plans in clean, noise-free settings. Despite solving fewer problems, the problems it does solve are addressed with higher-quality plans and greater validation reliability compared to the state-of-the-art. This behavior aligns with the demands of real-world scenarios, where safety and correctness often take precedence over coverage. In such contexts, it is more acceptable to refrain from planning than to produce invalid plans, making LAML's conservative bias a desirable feature.

These results highlight the importance of tailoring evaluation strategies to the operational context in which a learned planning domain will be used. They also suggest that the utility of a learned model cannot be fully captured by structural accuracy alone; practical performance under real-world conditions plays an equally important role.

The next chapter presents a reflective discussion of the key lessons and decisions made throughout the course of this thesis. It revisits the shift in perspective regarding the roles of behavior trees and planning domains, explores their representational differences, and addresses the challenges of interpretability and generalization. These reflections aim to situate the contributions of this work within the broader landscape of learning symbolic behavior models and provide guidance for future research directions.

Chapter 7

Discussion

The previous chapters have examined two complementary methods for representing the behavior of autonomous systems: Behavior Trees (BTs) and Planning Domains (PDs). Each of these representations has been studied in depth, both in terms of how it can be learned from data and how it supports planning and interpretability. This chapter adopts a different perspective. Rather than introducing new methods or results, it reflects on the methodological and conceptual choices that shaped the trajectory of the thesis. The purpose is to identify the key lessons learned during the course of this work, explain the rationale behind certain decisions, and offer a broader view of the implications for future research and industrial applications.

7.1 Topics for Discussion

This discussion is necessary for two reasons. First, several of the decisions taken during the development of this thesis represent a departure from the original research plan. As outlined in the introduction (Section 1.3), the initial approach was to learn planning domain representations, using BTs as an intermediate layer to structure and simplify the learning process. However, as the investigation progressed, it became increasingly clear that this assumption introduced new challenges rather than resolving existing ones, particularly regarding representational mismatch and the loss of procedural information. These challenges, which ultimately prompted a shift toward treating the learning of BTs and PDs as distinct problems, will be further examined in this chapter.

Second, this shift in perspective revealed important insights into the nature of the two representations themselves. BTs and PDs, although both intended to capture system behavior, differ in a fundamental way, not only in syntax and semantics, but also in their roles within autonomous systems. These differences

have direct consequences for the kinds of models that can be learned, the kinds of reasoning they support, and the extent to which they are amenable to human understanding and validation. It is precisely these consequences that motivate the present discussion.

Third, this thesis has been motivated from the outset by a concrete industrial context, namely Scania’s vision for autonomous fleet management. While much of the work has been conducted at the level of methodological and representational analysis, it is equally important to reflect on the extent to which the proposed methods can be applied in real-world scenarios.

The chapter is structured around four key reflections.

- Section 7.2 discusses the essential representational differences between BTs and PDs. While BTs model behavior as a reactive control structure with explicit execution flow, PDs describe a declarative model of action preconditions and effects. This divergence affects how goals are represented, how learning can be approached, and how each model scales to complex or multi-agent environments.
- Section 7.3 addresses the difficulties of translating from a learned BT to a generalizable PD representation. While the initial assumption was that BTs could provide a scaffold for inferring symbolic planning models, empirical findings and theoretical considerations revealed intrinsic obstacles to this approach. These challenges stem largely from the representational mismatch between procedural semantics (in BTs) and declarative semantics (in PDs), and they raise broader questions about the limits of hybrid modeling.
- Section 7.4 considers the issue of interpretability and human accessibility. One of the motivations for using symbolic representations is to support human understanding, debugging, and refinement. The work presented in this thesis has shown that BTs, when properly organized, can support these goals (see Section 4.4). By contrast, PDs, even when learned from well-structured data, pose significant challenges to human interpretability. This section explores the structural and semantic reasons for these differences and considers their implications for the design of interactive systems and tools.
- Finally, Section 7.5 returns to the industrial motivation, reflecting on the current distance to real-world applicability. It discusses the practical preconditions for testing the proposed methods in industrial contexts, considers proxy examples such as mining and hub-to-hub transport, and highlights the expected impact if the identified gaps in data can be addressed.

These reflections are not intended to offer final answers, but rather to highlight conceptual and practical challenges that emerged during the research.

They aim to contribute to a more nuanced understanding of how symbolic behavior models can be learned, interpreted, and applied in real-world autonomous systems.

7.2 Behavior Trees vs Planning Domains

Throughout the development of this thesis, it became increasingly clear that BTs and PDs are not simply two representations for the same kind of behavior model, but instead reflect fundamentally different assumptions about control, reasoning, and abstraction. This section examines the most significant representational contrasts, which are summarized in Table 7.1, and discusses how these differences shaped the learning approaches presented in earlier chapters.

Representation BTs are procedural structures that define behavior as a hierarchy of tasks and control nodes. In BTs, control flows from the root node down the tree, determined by node types (e.g., fallback, sequence) and runtime success/failure feedback. The tree structure explicitly encodes control flow logic, allowing designers to specify how an agent should act in different situations. Crucially, BTs do not model the state of the world explicitly. Instead, they rely on immediate execution feedback, such as action or condition success or failure, to guide behavior. Thus, the notion of *state* in BTs is implicit and procedural, emerging from the current status of nodes and the traversal logic of the tree itself.

By contrast, PDs adopt a declarative representation, typically formalized in languages such as the Planning Domain Definition Language (PDDL). In this paradigm, the *state* of the world is modeled explicitly as a set of symbolic variables or grounded predicates. Actions are defined by preconditions and effects, which describe transitions between states. PDs describe the building blocks from which plans that achieve a specified goal from a given initial state can be composed.

Therefore, BTs act as directly executable policies, whereas PDs require planning systems to synthesize behavior before execution. This distinction motivated the separation of learning strategies: while BTs can be inferred directly from observed execution sequences, learning a PD requires constructing an abstract model of environment dynamics

Reactivity BTs are designed to be reactive: they can re-evaluate and re-execute subtrees in response to failures or changing conditions. By contrast, classical planning approaches based on PDs typically involve compiling the domain model into a plan for a specific goal, given a specific initial state. If conditions change during execution, replanning is typically required, a process that can introduce latency and computational cost. While some planning frameworks incorporate plan repair or partial replanning, these techniques are

generally less responsive than the built-in reactivity of BTs. This difference shaped the evaluation methods in this thesis. In the case of PDs, evaluation focused on structural correctness and plan validity, using techniques such as comparison with historical plans, plan validation tools, and reference domain alignment. The emphasis was on whether the learned domain could reproduce or support the generation of plausible, executable plans under deterministic assumptions.

In contrast, the evaluation of BTs centered on runtime behavior and goal achievement, reflecting their reactive nature. Learned BTs were assessed using a fitness function that measured behavioral success in dynamic environments. The performance-based evaluation also captures the ability of BTs to adapt at runtime, which is not easily captured by static plan comparison or structural metrics.

Thus, the divergent assumptions about reactivity not only reflect different modeling philosophies but also necessitate distinct evaluation strategies aligned with the operational semantics of each representation.

Goal Handling BTs encode goals implicitly within their structure. They are tailored to execute specific behaviors and do not separate goal specification from execution logic. As a result, adapting BTs to new goals often requires structural redesign, as there is no notion of “goal satisfaction”.

Conversely, PDs are explicitly goal-driven. Goals are represented independently of the domain model, allowing the same set of actions to be used for a wide range of objectives. The planner synthesizes plans based on the difference between the current state and the goal state. This makes PDs more versatile in goal-driven contexts, but also more abstract and harder to validate.

Expressivity BTs are effective in constrained or well-defined environments but lack the representational expressivity required for complex, object-rich, or multi-agent domains. They do not support object parameterization or instantiation, meaning each agent usually has its own copy of the tree, and coordination must be manually engineered. PDs, on the other hand, support object-centric modeling through parameterized action schemas and object instantiation. A single domain model can represent a fleet of agents by instantiating different objects and reasoning over their interactions during plan generation. This enables centralized or distributed planners to coordinate multiple agents more efficiently. As such, PDs are more suitable for large-scale and heterogeneous environments, whereas BTs are best suited for individual agents with fixed roles and behaviors.

This difference proved to be one of the most significant practical challenges encountered during the thesis. In particular, it raised a fundamental modeling question: in multi-agent scenarios, should one learn a separate BT for each agent, or a single BT that captures the behavior of the entire system? The

former approach leads to duplication and overlooks shared structure, while the latter quickly becomes unmanageable in size and complexity. To mitigate this, it became necessary to explore techniques for generalization and abstraction, a form of “lifting” for BTs, that could encode parameterized behavior or capture repeated patterns across agents.

This tension between modularity and generality, and the lack of native parameterization in BTs, highlighted a key limitation of the formalism when applied to domains beyond single-agent, fixed-role tasks. These insights directly informed the decision to treat the learning of BTs and PDs as two distinct challenges, and they motivated the pursuit of lifted representations in the context of planning domain learning.

Interpretability A recurring theme in this thesis has been human verification, validation, and debugging. As discussed in Section 4.4, BTs are generally considered interpretable due to their explicit control flow and modular, graphical structure. The user study confirms this assumption, but with important qualifications. Interpretability was not uniformly observed across all tree structures. Rather, it was contingent on specific design choices, BT structure, and user characteristics. Nevertheless, BTs appear to require minimal learning effort: users with no prior knowledge performed well on compact, well-structured trees, reinforcing BTs’ accessibility. In contrast, PDs, though formally well-defined, require familiarity with planning languages such as PDDL, resulting in a steeper learning curve and reduced interpretability for users without a background in automated planning.

However, in multi-agent scenarios, BTs face a significant scalability challenge. Representing separate trees per agent can lead to redundancy, while consolidating multiple behaviors into a single tree risks combinatorial growth and loss of clarity. PDs, despite their abstraction, provide a parameterized and scalable representation more appropriate for such domains, though this comes at the cost of reduced transparency and accessibility.

These trade-offs between interpretability and scalability contributed directly to a key methodological decision in this thesis: to pursue the learning of BTs and PDs as distinct and complementary problems. While BTs support intuitive, human-facing models of system behavior, PDs offer generality and abstraction necessary for complex, multi-agent planning, highlighting the need for dedicated learning strategies and evaluation criteria for each representation.

7.3 Converting Behavior Trees to Planning Domains

This section revisits the first methodological approach introduced in Section 1.3.1, which proposed to extract planning domain models from existing

Table 7.1: Comparison between Behavior Tree (BT) and Planning Domain (PD) representations

Aspect	BT	PDs
Representation	Procedural; behavior encoded as tree structures with explicit control flow; state is implicit via node status	Declarative; domain dynamics defined using formal languages (e.g., PDDL); state is explicitly modeled
Reactivity	High; supports runtime adaptation to failures or changing conditions	Limited; typically requires replanning
Goal Handling	Goals are embedded in the structure; adapting to new goals often requires modifying or redesigning the tree	Goals are specified independently from the domain; the same domain can support a wide range of goals through plan generation
Expressivity	Low; no native support for object parameterization or multi-agent abstraction	High; supports parameterized actions and reasoning over many agents or objects
Interpretability	Generally high due to graphical structure and explicit control flow, though unbalanced trees may be complex	Varies; individual operators are interpretable, but plans can be opaque

BTs. The underlying motivation was to leverage the structured procedural knowledge encoded in BTs as a source of prior information for planning domain learning. Since BTs explicitly encode control flow and task structure, it was initially hypothesized that a suitable transformation could yield a declarative planning domain capturing the agent’s behavior in a form amenable to classical planning.

Such a conversion appeared promising for two reasons. First, BTs are commonly used in industrial and robotic applications, meaning they could serve as a rich, interpretable source of expert behavior. Second, reusing BTs to derive planning models would allow bootstrapping planning-based approaches without starting from raw data or manually crafted domain models.

Significant conceptual and technical obstacles make the transformation from BT to PD inherently non-trivial, due to fundamental mismatches in rep-

resentation paradigms and modeling assumptions. These difficulties prompted a shift in the approach: rather than converting from BTs to PDs, the revised approach focused on learning PDs directly from execution traces, while using BTs as a complementary representation to support interpretability and human-in-the-loop validation.

The remainder of this section elaborates on the challenges encountered during the attempted conversion and situates them within the broader literature on BT-PD transformations.

Challenges of Conversion

Transitioning from BTs to PDs involves converting the procedural, hierarchical constructs of BTs into the declarative, state-based representations characteristic of PDs. Furthermore, BTs encode goals within their structure, whereas PDs separate goal specification from execution logic. This structural difference necessitates rethinking goal representation during conversion [139]. A deeper challenge arises when attempting to extract the action models required for planning. While preconditions can sometimes be inferred by examining the structure of the BT (e.g., what must succeed before an action is executed), postconditions are rarely made explicit. BTs typically encode success or failure outcomes but omit detailed state change information. This omission makes it extremely difficult, and in many cases infeasible, to infer the declarative effects needed to define planning operators, a challenge that reflects the broader difficulty of extracting explicit causal relations from data [14]. Without explicit annotations or external semantic models, there is no reliable basis for deriving how a successful action transforms the world state. This asymmetry between preconditions and postconditions poses a fundamental obstacle to converting BTs into PDs.

Another significant challenge lies in the representational mismatch. BTs are often designed for a specific scenario, with actions applied to concrete entities or locations. This propositional encoding is inherently limited in scope and generality. In contrast, PDs are designed to operate over lifted action schemas, using variables that can be instantiated to different objects, agents, or roles at planning time. As such, a direct translation from a BT to a PD often yields only a single, grounded instance of behavior, rather than a generalizable model. This limits the scalability and reuse of the resulting domain and undermines one of the key advantages of planning frameworks: their ability to reason over multiple agents and varying contexts through variable binding and parameterization.

While converting a BT into a PD presents considerable challenges, the inverse process, deriving BTs from planning-based representations, has been more extensively studied. One line of research, presented by Martin et al. [139], proposes systematic algorithms to transform PDDL action plans into equivalent BT structures, preserving the logical execution order while enabling reactivity during runtime. In the same vein, Zapf et al. [182] introduce formal

algorithms for converting temporal plans into BTs that respect time-based constraints, ensuring correct sequencing and synchronization in domains where temporal requirements are critical. In the domain of game AI, hybrid frameworks have emerged in which Hierarchical Task Network (HTN) planners generate task decompositions that are subsequently mapped onto BTs. Neufeld et al. [120] replace the lowest-level HTN methods with BTs that execute the corresponding tasks, noting that HTN decompositions map naturally to BT subtrees (e.g., ordered HTN subtasks map to BT sequence nodes). Also in task and motion planning (TAMP), integrated pipelines have been developed by Verma et al. [169] to convert a plan into a BT. Further research has focused on optimizing the construction of BTs from planning models by leveraging control-flow patterns to enable parallel execution and increased runtime adaptability [22].

Hybrid architectures combining BTs and PDs have also been studied. For instance, a two-tiered system has been proposed by Liu et al. [103] in which a PDDL planner decomposes high-level goals into intermediate tasks, while individual BTs executed on each robot carry out these tasks and monitor for failures.

Asymmetry and Expressiveness

These studies demonstrate that the transformation from planning models to BTs has proven to be a productive and versatile strategy across a variety of domains. However, the reverse process, from BT to PD, remains significantly more complex. This asymmetry arises, in part, from fundamental differences in expressiveness. Conceptually, BTs can be seen as representing a more constrained subset of the behaviors that PDs can express. While it is often possible to compile a PD into a corresponding BT, albeit sometimes with increased complexity or reduced generality, the reverse is not true. BTs are limited in their ability to capture the abstract, goal-directed reasoning and flexible object manipulation that planning domain models support. As a result, certain high-level strategies and generalizations inherent to planning frameworks cannot be fully captured within the BT formalism.

7.4 Humans and Planning Domains

The development and maintenance of planning domains remain a challenging task, particularly when considered in real-world contexts where domain complexity and operational variability are significant. Despite the extensive formalism and precision underlying planning languages such as PDDL, human involvement in the validation and debugging of these domains is both indispensable and profoundly limited [104]. These limitations likely stem from cognitive, representational, and methodological mismatches between human

reasoning and the abstract, formal models employed in automated planning systems. At the core of this difficulty is the symbolic and highly formal nature of planning domains. These models require domain engineers to explicitly encode all relevant actions, preconditions, and effects that capture the dynamics of the environment. However, such representations do not align naturally with the way humans conceptualize tasks or domain behavior. As a result, errors in model specification, whether due to omissions, misrepresentations, or incorrect causal structures, often go unnoticed until they manifest as planning failures.

Challenges of Human Validation

Validation of planning domains typically involves ensuring that a domain accurately captures the intended behavior and that generated plans are feasible and appropriate. However, this task becomes increasingly difficult as the complexity of the domain increases. The combinatorial nature of planning models implies that even minor changes in preconditions or effects can lead to significant shifts in plan structure, which may not be easily anticipated by human designers.

Another complicating factor is the general lack of intuitive feedback mechanisms, such as visual or semantic representations of planning logic, which further hinders effective human debugging. Although simulators for plan trace visualization exist, they often assume a high level of familiarity with planning formalisms, limiting their accessibility to non-expert users or domain stakeholders. Vaquero et al. [168] present an extensive study of planning knowledge engineering tools and approaches. It is noteworthy that there is a multitude of tools aimed at verification of PDDL, such as VAL [73] or PDver [137]. However, at some point in the knowledge engineering workflow, direct inspection and manual editing of PDDL remain necessary. The challenges associated with this process have driven interest in developing human-in-the-loop frameworks that enable systematic, iterative refinement of plans. Rather than viewing domain modeling as a static, front-loaded process, these approaches emphasize continuous human engagement throughout the planning pipeline.

Human-in-the-Loop Approaches

A pivotal conceptual shift in this direction was introduced by Kambhampati [89] through the notion of model-lite planning, which reframed the relationship between humans and AI in domain modeling. Rather than requiring a complete and correct domain model upfront, model-lite planning tolerates incompleteness and allows missing components to be resolved dynamically through interaction with a human or external knowledge sources. This paradigm positions human users as active collaborators during planning execution, supplying missing preconditions, clarifying intended effects, or revising the model as needed.

Several frameworks have been proposed to more tightly integrate human feedback into planning workflows, such as RADAR [153] that incorporates automated planners within decision-making processes to assist human experts in refining and executing plans. In the area of mixed-initiative planning, Cox & Zhang [32] enable users to influence and steer goal formulations to improve plan quality. Collaborative planning approaches, exemplified by Kim et al. [92], allow humans to supply high-level guidance in the form of soft preferences to shape the planner’s low-level search. Finally, Kulkarni et al. [97] explore planning support in environments where humans and AI agents coexist, focusing on synthesizing proactive assistance behaviors that reduce task costs while remaining transparent to the human.

More recently, the advent of Large Language Models (LLMs) in planning has opened a new frontier for human-in-the-loop. LLMs can generate candidate PDDL models from high-level descriptions or even auto-complete parts of a domain. However, as recent studies highlight [62, 58], LLM-generated domains often suffer from structural and semantic inconsistencies, such as undefined predicates, logically incoherent actions, or unrealistic causal chains. Moreover, LLMs do not inherently reason about domain correctness; they generate syntactically plausible outputs, but lack the capability to explain or validate domain behavior. Consequently, human intervention remains necessary to review, correct, and ensure the validity of the resulting models. All of these approaches aim to reduce the cognitive burden on users by allowing them to express constraints, preferences, or high-level guidance in more natural and intuitive ways. While they represent significant progress in human-AI collaboration for planning, most continue to focus primarily on plan-level interaction, rather than enabling robust human oversight of the domain model itself. As a result, domain representations often remain opaque and difficult to inspect or refine manually, particularly for non-expert users.

In response to this persistent transparency problem, Fox et al. [45] introduced the field of explainable planning, framing it as a means to provide human understandable, context-sensitive explanations of planner behavior and decision making. Explainable planning methods improve interpretability at the level of individual plans by explaining why a given plan was chosen, what alternatives were considered, or why a goal is unreachable. However, they do not fully address the validation of the underlying domain model, nor do they offer mechanisms for guiding humans in systematically debugging or correcting domain errors. As such, the gap between improving plan comprehension and enabling effective domain model validation remains.

Hence, while human involvement remains essential in the development and oversight of planning domains, the current methods and tools fall short in supporting effective human validation and debugging. The symbolic, abstract nature of planning models, combined with the lack of intuitive interfaces and real-world variability, poses significant cognitive and practical challenges. Future work should aim to better integrate domain learning, visualization, and

explainability techniques in ways that align with human reasoning processes. This includes developing domain-level feedback systems, integrating naturalistic model representations, and fostering cross-disciplinary collaborations that draw from human-computer interaction, cognitive systems engineering, and formal methods. Addressing these gaps is crucial for the reliable deployment of planning systems in complex, real-world settings.

7.5 Applicability to Real-World Scenarios

This thesis has been motivated by Scania’s vision for intelligent offboard decision-making in autonomous fleet orchestration (Section 1.1.2). It is therefore important to reflect on how the proposed contributions could be applied in real-world scenarios, what conditions would be required for such an application, and what impact could be expected once these conditions are met.

Challenges and Proxy Scenarios

At present, applying the developed methods directly to Scania’s industrial settings is not feasible. The main obstacle is the lack of sufficiently detailed execution traces. Although large volumes of operational data are produced in industrial contexts, they are rarely available in a form suitable for learning symbolic behavior models. What is typically accessible are low-level logs, which require substantial post-processing in order to obtain cleansed and conformed data. In particular, what is missing are clear mappings between task-level actions and system states, as well as time-stamped action labels, and snapshots of relevant state variables before and after execution. Without this level of detail, the learning methods cannot be meaningfully applied to real-world traces.

It is important to emphasise, however, that this limitation is not inherent to the methods themselves but rather a matter of data availability and analysis. If richer execution traces could be collected, the contributions presented here would become directly testable. In such a setting, Behavior Tree learning through BT-Factor could be used to generate compact and interpretable structures for vehicle-level tasks such as loading, unloading, or dispatch, while planning domain learning through LAML could support higher-level fleet coordination, for example, in resource allocation. In detail, the LAML approach explicitly accounts for noise in the data, which means that industrial traces would not need to be perfectly clean. Noisy or partially inconsistent logs could still be exploited to infer robust models, provided the essential structural information is present.

Instead of real-world traces, this thesis has relied on illustrative scenarios that serve as proxies for real-world problems. The simulation experiments used to evaluate BT-Factor (Section 3.3.1) reproduce the dynamics of a logistics hub, where an autonomous agent operates across multiple locations. This

scenario is inspired by Scania’s autonomous mining applications¹, where fleets of trucks must be coordinated across interdependent tasks such as loading, hauling, and dumping under resource constraints. In such a context, learned BTs can capture robust and compact local behaviors, while planning domains can coordinate multi-vehicle allocation, for instance, sequencing departures under changing demands or rerouting vehicles in response to delays.

A further source of insight comes from the International Planning Competition (IPC) domains, which provide abstracted but challenging environments in which planning domain learning is usually tested. For example, the Driver-Log domain, used in the Third IPC (2002) and applied here to evaluate the LAML method, involves transporting packages between locations using trucks and drivers navigating road networks. This benchmark closely resembles Scania’s long-haul hub-to-hub logistics scenario², where autonomous trucks operate continuously between hubs. In both cases, the challenge lies in coordinating vehicles across a network under constraints, making the domain an effective testbed for developing noise-tolerant planning models.

Path to Deployment and Impact

Taken together, these reflections highlight both the promise and the distance to real-world applicability. The core contributions demonstrate that it is possible to learn structured and interpretable behavior models from execution data, and that such learning can tolerate noise and imperfections in the data. To realise their industrial potential, however, targeted steps are required in data collection and preprocessing. If these conditions are met, the picture changes substantially: planning for autonomous fleets can shift from handcrafted, static models toward data-driven, auditable representations that combine operational efficiency with human transparency.

Reflecting on these examples also clarifies what a plausible path to deployment would look like. The first step would be the definition of a *logging contract* in which a core set of predicates, actions, and event outcomes are systematically recorded during operations. Once such a schema is in place, an initial dataset could be collected over a representative period, capturing both routine operations and typical disturbances.

This would provide the input for running the learning pipeline: BTs could be extracted and compacted through BT-Factor, while planning domains could be learned and validated under noise-tolerant assumptions using LAML. Human operators would then review the resulting models, supplying missing causal links or correcting mis-specified effects, before the models are tested in simulation or shadow mode alongside baseline heuristics. In this way, each

¹<https://www.scania.com/group/en/home/innovation/autonomous-solutions/mining.html>

²<https://www.scania.com/group/en/home/innovation/autonomous-solutions/hub-to-hub.html>

stage of deployment (data collection, model learning, human validation, and simulated trials) builds constructively towards eventual use in live operations.

From today's perspective, it is clear that several gaps still separate this work from real-world applications. The main barriers are the absence of standardised logging, the limited visibility of key fleet-level states, and the lack of an integrated simulation environment for testing. Data sparsity and shifts in operating conditions present further risks, since rare events and changing operational contexts may undermine generalization. Yet these limitations also suggest the concrete steps needed to move forward, namely improving data collection, enriching trace annotations, and developing reliable environments in which learned models can be simulated and tested. With these conditions in place, the methods presented in this thesis could be validated end-to-end in realistic, controlled trials.

The expected impact of such an integration is considerable. Learning from execution traces would enable planning to become both more efficient, through compact and reactive behavior Trees, and more applicable, through validated and noise-tolerant planning domains. For industry, this would mean a shift away from handcrafted, static models toward data-driven, auditable representations. For practitioners, it would provide tools that are transparent and open to inspection, thereby enhancing trust and accountability. Ultimately, if the outlined preconditions can be met, the contributions of this thesis would help bring industrial fleet management closer to realising Scania's vision of scalable, efficient, and safe autonomous transport.

Chapter 8

Conclusions

This thesis tackles the challenge of learning interpretable and actionable models of system behavior from execution traces in autonomous environments. The key contributions of this work include new methods for learning behavior representations from noisy and realistic data, a formal framework for human-in-the-loop validation and refinement, and evaluation strategies tailored to the specific properties of different representations. Together, these contributions address core challenges in learning behavior representations from real-world data, an area where existing approaches often rely on handcrafted representations or fail to support robust human oversight. We address the challenges by reducing reliance on manual domain engineering, supporting user interaction during model refinement, and introducing targeted evaluation techniques.

The remainder of this chapter revisits the four research questions that guided the thesis, outlining the main contributions associated with each. It then discusses the limitations and assumptions of the approach, reflects on the ethical, social, and economic implications, and concludes with directions for future research.

8.1 Summary of the Contributions

The central goal of this thesis was to explore how symbolic representations of system behavior, specifically action models, can be learned from execution traces in real-world environments. In doing so, it addressed four core research questions, each representing a critical aspect of the learning pipeline: from data acquisition to representation, validation, and evaluation. In this section, each contribution is explicitly tied to the corresponding research question(s) and contextualized in terms of its novelty and significance. Taken together, these contributions address the dual requirement of action representations, as originally framed by the knowledge representation hypothesis [15]: they are

designed to support automated reasoning while remaining accessible to human understanding and validation.

RQ1: How can interpretable representations of system behavior be learned from execution traces in autonomous environments?

This question has been addressed through two complementary methods presented in Chapter 3 and Chapter 5. First, a novel approach was introduced for learning Behavior Trees (BTs) by combining circuit theory with decision tree induction techniques. This method enables the automatic generation of models of behavior that are both expressive and compact. Compared to prior approaches that treat BTs as handcrafted or heuristically constructed artifacts, this contribution provides a data-driven, principled framework for their compact synthesis from execution logs.

Second, a method was developed for learning planning domain models, in the form of PDDL operators, from execution traces. This method is tailored to operate under realistic conditions that include sensor noise. It advances the field by demonstrating the feasibility of learning symbolic planning models directly from raw logs, thereby reducing the reliance on hand-engineered domain knowledge. By incorporating techniques such as decision tree learning on state variables, the approach ensures robustness to noise.

These two methods enable the automatic induction of behavior representations from raw execution data, even in the presence of noise and without prior domain knowledge, thereby addressing RQ1. By bridging the gap between low-level observations and high-level models, they make it possible to derive structured, interpretable representations that can be directly used for analysis, explanation, or planning, while differing in their assumptions and applicability as discussed in Chapter 7. This opens up new opportunities for applying automated planning in domains where manual modeling would be infeasible, and lays the groundwork for integrating learned behavior models into broader autonomous system architectures.

RQ2: What forms of representation best support interpretability and facilitate human understanding of system behavior?

Two principal contributions address this question by advancing the interpretability of symbolic representations. Chapter 1 introduces a framework for human-in-the-loop validation, debugging, and refinement of learned behavior models. This framework formalizes the ways in which users, such as engineers, operators, or domain experts, can interact with symbolic models to inspect, correct, or guide their evolution. Chapter 4 reports findings from a user study investigating the interpretability of BTs.

These two contributions together offer both a conceptual foundation and empirical support for the role of BTs as human-aligned representations of au-

tonomous behavior, and therefore propose BTs as a justified answer to the question posed in RQ2.

By identifying BTs as both operationally effective and cognitively accessible, these contributions clarify the design space for representations that can be inspected, edited, and trusted by human users. This enables a meaningful connection between system transparency and usability, offering direct benefits to those who develop, deploy, or oversee autonomous systems.

RQ3: How can human users be effectively integrated into the process of validating, correcting, and refining learned representations?

This question is addressed through the aforementioned framework for human-in-the-loop interaction with symbolic behavior models, presented in Chapter 1. Unlike previous approaches that incorporate human feedback in an ad hoc or informal manner, this framework provides structured interfaces and workflows that guide user involvement in a principled way.

A mixed-initiative paradigm has been proposed in which model correctness and usability emerge from collaboration between human and machine, extending the scope of symbolic learning beyond automation.

By formalizing the space of possible user interactions and demonstrating their effect on model improvement, this contribution advances the field's understanding of how human expertise can be systematically integrated into symbolic learning pipelines, providing an answer to RQ3.

This work creates the conditions for integrating domain experts more effectively into the model development cycle, not as passive evaluators, but as active participants in shaping system behavior. By shifting their role from low-level specification to higher-level guidance and oversight, the framework allows organizations to continue leveraging expert knowledge while increasing the efficiency, consistency, and scalability of symbolic model development.

RQ4: What metrics and methods can be used to evaluate specific properties of the learned representations?

Two evaluation frameworks have been introduced in Chapter 4 and Chapter 6, tailored to different types of symbolic representations: one for BTs, and one for learned planning domain models.

For BTs, a structured evaluation guide has been developed that includes a set of design principles, metrics, and properties that support both functional performance and human usability. This framework enables systematic comparison between BTs, helps identify structural weaknesses, and supports iterative refinement during model development. It fills a gap in the literature by offering a comprehensive and interpretable set of evaluation criteria beyond task success or execution efficiency.

For learned planning domain models, a systematic evaluation framework has been presented that combines quantitative metrics (e.g., precision and recall) with qualitative assessments (e.g., alignment with expert models). This dual approach ensures that learned models are not only functionally sound but also interpretable and trustworthy. The framework also incorporates different evaluation techniques, such as model-based simulation and expert review, to validate symbolic accuracy under real-world constraints.

These evaluation contributions provide the field with robust tools to assess the correctness, usefulness, and interpretability of learned symbolic models in both academic and operational settings, thus answering RQ4.

By providing representation-specific evaluation strategies that go beyond task success, it is possible to systematically assess the reliability and usability of learned symbolic models. These methods benefit both researchers and practitioners by enabling more rigorous model validation, facilitating human understanding, and supporting continuous improvement in safety-critical systems.

8.2 Industrial Relevance

This thesis was originally motivated by an industrial need identified in collaboration with Scania, which envisions autonomous transport systems that extend beyond onboard autonomy to include intelligent offboard decision-making for fleet orchestration and logistics coordination. The methods developed here, particularly for learning interpretable behavior models and integrating human oversight, lay critical groundwork toward this vision by enabling data-driven, modular, and inspectable representations of system behavior. These contributions support the automation of planning and decision-making tasks in dynamic and resource-constrained environments.

However, to fully realize Scania’s vision, further developments are required. As highlighted in Section 7.5, real-world deployment will require access to larger volumes of high-quality execution data, ideally collected across diverse vehicle types, operational conditions, and task scenarios. Second, the current methods operate at a mid-level of abstraction; for practical integration with fleet-level decision-making, they must be extended to capture higher-level planning logic, such as task interdependencies, resource allocation constraints, and strategic scheduling objectives. Finally, the robustness of learned models under partial observability, asynchronous data, and evolving environments must be further developed, alongside seamless integration with existing planning and operations infrastructure. Addressing these gaps is essential for transitioning from proof-of-concept models to deployable decision-support systems in autonomous fleet management.

8.3 Limitations and Assumptions

While the contributions of this thesis advance the state of the art in learning interpretable behavior representations for autonomous systems, several limitations and assumptions must be acknowledged. Each presents both a constraint on the current work and a potential direction for future research.

Dependence on high-quality, temporally annotated execution traces

The learning methods assume access to structured, time-ordered traces that reflect system execution over time. In practice, such traces may be incomplete or inconsistently logged, particularly in legacy systems or those with limited instrumentation. While this assumption is reasonable in well-instrumented development settings, it presents a barrier in operational or resource-constrained environments. Relaxing this assumption would require developing methods that can reconstruct or infer missing temporal structure. There is active work in this area in process mining [37, 131] communities, which could offer promising foundations for generalization.

Assumptions of deterministic dynamics and full observability

This thesis assumes that the system operates under deterministic dynamics and that the execution traces provide full observability over relevant state variables. This does not imply that the observed system is fully deterministic in practice; rather, it is assumed that non-deterministic effects, such as those arising from stochastic processes, environmental variability, or unmodeled interactions, manifest as observational noise. Under this interpretation, the learning problem is framed as recovering a deterministic model that best approximates the system's behavior, with non-determinism treated as irregularities in the data rather than as intrinsic features of the system model.

Accordingly, the learning methods are designed to tolerate sensor noise and imperfect data, and have been validated under such conditions. However, the assumption remains that all the key aspects of system state are, at least in principle, observable: there is no support for latent state inference or reasoning under occlusion or missing-state dimensions.

In practical applications, this assumption is moderately limiting. Many real-world systems operate under partial observability due to occluded sensors, asynchronous data streams, or internal system variables that are not externally visible. Relaxing this assumption would require new methods that infer latent states, perform belief tracking, or operate over abstracted representations learned from incomplete data. While the challenge of planning in partially observable stochastic domains has been investigated for several

decades [88], and recent work has begun to explore behavior model learning in such settings [87], the development of interpretable models that support human-in-the-loop validation in these contexts remains an open and significant research challenge.

Limited empirical validation of interpretability claims

The thesis includes a user study and qualitative analyses to evaluate the interpretability of BTs, but the scale and diversity of participants and tasks were limited. While initial results are promising, more extensive empirical work is needed to generalize findings across domains, user expertise levels, and representation types. In practical terms, this is a moderate limitation: the proposed models are interpretable by design, but their usability in collaborative, safety-critical, or regulated environments has yet to be fully validated. Addressing this would require large-scale human-subject evaluations, potentially including comparative studies with alternative representations such as finite state machines, or measuring trust, task accuracy, and error detection. Insights from human-computer interaction (HCI) and cognitive science could guide the design of such evaluations.

Preliminary state of the human-in-the-loop refinement framework

The human-in-the-loop approach proposed in this thesis offers structured ways for users to validate and refine learned models, but the framework remains an early prototype. It currently lacks formal modeling of cognitive load, error recovery, or task-specific guidance. This is a significant limitation for deployment in real-world settings where users must interact with learning systems under time pressure or with limited training. Overcoming this would involve designing and evaluating user-centered interaction models, potentially drawing from explainable AI [121] or mixed-initiative planning [90] literature. Future work could explore interface design patterns, adaptation to user expertise, or even collaborative learning protocols that explicitly balance machine and human contributions.

8.4 Ethical, Social, Economic impact

This thesis contributes to the responsible development of AI systems by focusing on transparency, interpretability, and human oversight. These concerns align closely with the ethical requirements outlined in the European Commission’s *Ethics Guidelines for Trustworthy AI* (2019) [70], which emphasize the importance of human agency, technical robustness, and explicability in AI systems.

From an ethical standpoint, the emphasis on human-in-the-loop validation supports accountability and mitigates risks associated with autonomous decision-making. A key concern, however, is the dependence of the proposed methods on the quality and provenance of execution logs. If these logs reflect biased or discriminatory practices, then the learned models may inadvertently reproduce and legitimise such patterns. By allowing domain experts to inspect and influence symbolic models, the approach reduces the likelihood of opaque or misaligned system behavior. This work also contributes to the broader goal of value alignment, ensuring that AI systems act in accordance with human intentions and societal norms [144].

Socially, the ability to learn and validate interpretable representations opens new possibilities for transparency in the operation of autonomous systems. This responds to growing public and institutional demands for algorithmic accountability and explainability. The capacity to justify AI behavior is essential not only for fostering user trust but also for meeting regulatory requirements and supporting democratic oversight. By producing structured, symbolic models that can be inspected and revised by humans, this work promotes the socially responsible deployment of AI in safety-critical and ethically sensitive domains. Crucially, the emphasis on interpretability is not intended to replace human operators, but to transform their role: instead of engaging in low-level, manual specification of behavior, human experts are repositioned as supervisors and decision-makers who guide, monitor, and refine system behavior. Their domain expertise remains essential, but is leveraged at a higher level of abstraction, enabling more effective and accountable integration of AI into human-centered workflows.

From an economic standpoint, the proposed methods reduce the reliance on manually specified behavior models, lowering the development costs associated with deploying AI systems in complex environments. By enabling automated yet interpretable model acquisition, the approach supports greater scalability and responsiveness to change, which is particularly valuable in dynamic or resource-constrained domains. Moreover, rather than displacing human labor, the approach redistributes expertise: human operators are transitioned into roles that emphasize oversight, strategic guidance, and model auditing. This not only preserves domain knowledge but also creates economically sustainable roles that combine technical supervision with decision-making authority. As such, the methods proposed here offer a pathway toward economically efficient, yet human-centered, AI integration.

More broadly, this work situates itself within ongoing debates about the balance between automation and human control in AI system design. It offers an alternative to end-to-end black-box learning by advancing methods that foreground symbolic structure, traceability, and human involvement. In doing so, it contributes to a growing body of research advocating for AI systems that are not only intelligent and efficient, but also interpretable, corrigible, and aligned with human values.

8.5 Future Work

This thesis lays a foundation for learning and validating symbolic models of system behavior in real-world environments. Nevertheless, as discussed in Section 8.3, several avenues for future research remain open, both to extend the theoretical contributions and to enhance the practical applicability of the proposed methods.

Regardless of limitations and assumptions, an extension of this work concerns the development of continual and online learning techniques for symbolic behavior representations. The methods proposed in this thesis operate under a batch learning paradigm, where models are induced from complete execution traces collected offline. While effective for initial model acquisition, this setting can be insufficient for real-world deployments where systems operate over extended periods, undergo updates, and encounter novel situations. In such contexts, the ability to incrementally revise the learned representations based on newly observed behavior, without retraining from scratch or forgetting prior knowledge, becomes essential. The techniques introduced in this thesis offer a promising basis for such extensions. In particular, the operator learning method is structured around variable-wise analysis of preconditions, which supports modularity and localized revision. When previously unseen state variables are encountered, the learning process could be selectively reapplied to those variables. This property facilitates efficient adaptation while preserving existing knowledge.

There is also significant potential for expanding the human-in-the-loop framework introduced in this thesis. The current approach formalizes mechanisms for symbolic model inspection and correction, assuming a domain expert capable of interacting with representations such as BTs or planning domains. Building on this foundation, future research could investigate adaptive and personalized interaction strategies that better accommodate variation in user expertise, cognitive load, and task complexity. For example, the interpretability and modular structure of the learned models developed in this thesis could support the design of interfaces that dynamically tailor the level of detail presented, or prioritize specific components for user attention based on past correction patterns. While the current implementation assumes a static interaction protocol, extensions could introduce adaptive elements such as explanation-based feedback, natural language querying, or visual analytics to enhance accessibility. Furthermore, incorporating personalization mechanisms, where the system learns from user interactions over time, would require relaxing current assumptions about one-off model validation and instead supporting iterative, collaborative refinement. Such directions build on the transparency and editability established in this work, while requiring new methods for feedback integration and adaptive interface design.

Another critical area for future exploration is large-scale evaluation and benchmarking. While this thesis proposes principled evaluation frameworks

and reports empirical findings, broader validation across domains and user populations is needed to generalize the results. Future work could include the creation of standardized datasets, executed traces, and benchmark tasks for behavior learning.

Finally, if behavior representations are to be used in safety-critical applications, the need for formal verification and safety analysis becomes increasingly important. Ensuring that learned models satisfy correctness constraints, support goal achievement, and avoid unintended behavior is a non-trivial challenge. Future research could explore how to verify structural properties of behavior models, assess robustness under perturbations, and integrate behavior learning with formal methods for safety assurance. Such developments would be essential for the trustworthy deployment of symbolic AI in domains such as autonomous vehicles, robotics, and critical infrastructure.

References

- [1] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1990. (Cited on pages 59, 60, and 62.)
- [2] Don Joven Agravante, Daiki Kimura, Michiaki Tatsubori, Asim Munawar, and Alexander Gray. Learning neuro-symbolic world models with conversational proprioception. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 648–656, 2023. (Cited on page 85.)
- [3] Alper Ahmetoglu, Erhan Oztop, and Emre Ugur. Symbolic manipulation planning with discovered object and relational predicates. *IEEE Robotics and Automation Letters*, pages 1968–1975, 2025. (Cited on page 84.)
- [4] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, pages 104–137, 2019. (Cited on pages 83 and 103.)
- [5] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, pages 349–402, 2008. (Cited on page 83.)
- [6] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, pages 469–483, 2009. (Cited on page 34.)
- [7] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 2018. (Cited on pages 2 and 82.)
- [8] Masataro Asai, Hiroshi Kajino, Alex Fukunaga, and Christian Muise. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, pages 1599–1686, 2022. (Cited on page 84.)
- [9] Tomáš Balyo, Martin Suda, Lukáš Chrupa, Dominik Šafránek, Stephan Gocht, Filip Dvořák, Roman Barták, and G Michael Youngblood. Planning domain model acquisition from state traces without action parameters. *arXiv preprint arXiv:2402.10726*, 2024. (Cited on page 83.)

- [10] Roman Barták. Visualizing plans. In Marco Vallati and Daniele Magazzeni, editors, *The Role of AI Planning in Robotics and Autonomous Systems*, pages 157–172. Springer, 2020. (Cited on page 2.)
- [11] Scott Sherwood Benson. *Learning action models for reactive autonomous agents*. Stanford University, 1997. (Cited on page 82.)
- [12] Oliver Biggar, Mohammad Zamani, and Iman Shames. A principled analysis of behavior trees and their generalisations, 2020. (Cited on pages 23 and 58.)
- [13] Oliver Biggar, Mohammad Zamani, and Iman Shames. An expressiveness hierarchy of behavior trees and related architectures. *IEEE Robotics and Automation Letters*, pages 5397–5404, 2021. (Cited on page 54.)
- [14] Eva Blomqvist, Marjan Alirezaie, and Marina Santini. Towards causal knowledge graphs - position paper. In *In Proceedings of Workshop on The Knowledge Discovery in Healthcare Data (KDH)@ ECAI*, 2020. (Cited on page 121.)
- [15] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2004. (Cited on pages 2 and 129.)
- [16] Robert K. Brayton, Gary D. Hachtel, Curtis McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic minimization algorithms for VLSI synthesis*. Springer Science & Business Media, 1984. (Cited on pages 36 and 39.)
- [17] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. Classification and regression trees. *Wadsworth International Group*, pages 452–456, 1984. (Cited on pages 16 and 35.)
- [18] Daniel Bryce and Olivier Buffet. International planning competition uncertainty part: Benchmarks and results. In *The Sixth International Planning Competition, ICAPS*, 2008. (Cited on page 31.)
- [19] Zhongxuan Cai, Minglong Li, Wanrong Huang, and Wenjing Yang. Bt expansion: a sound and complete algorithm for behavior planning of intelligent robots with behavior trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 6058–6065, 2021. (Cited on pages 60, 64, 66, and 68.)
- [20] Lina Castano and Huan Xu. Safe decision making for risk mitigation of uas. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1326–1335. IEEE, 2019. (Cited on page 64.)

- [21] Lukáš Chrpa, Carmine Dodaro, Marco Maratea, Marco Mochi, and Mauro Vallati. Comparing planning domain models using answer set programming. In *European Conference on Logics in Artificial Intelligence*, pages 227–242. Springer, 2023. (Cited on page 101.)
- [22] Michele Colledanchise, Diogo Almeida, and Petter Ögren. Towards blended reactive planning and acting using behavior trees. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8839–8845. IEEE, 2019. (Cited on pages 68 and 122.)
- [23] Michele Colledanchise, Giuseppe Cicala, Daniele E. Domenichelli, Lorenzo Natale, and Armando Tacchella. Formalizing the execution context of behavior trees for runtime verification of deliberative policies. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9841–9848. IEEE, 2021. (Cited on pages 60 and 64.)
- [24] Michele Colledanchise, Alejandro Marzinotto, and Petter Ögren. Performance analysis of stochastic behavior trees. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3265–3272. IEEE, 2014. (Cited on pages 23 and 55.)
- [25] Michele Colledanchise and Lorenzo Natale. Analysis and exploitation of synchronized parallel executions in behavior trees. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6399–6406. IEEE, 2019. (Cited on page 55.)
- [26] Michele Colledanchise and Lorenzo Natale. Handling concurrency in behavior trees. *IEEE Transactions on Robotics*, pages 2557–2576, 2021. (Cited on page 55.)
- [27] Michele Colledanchise and Petter Ögren. How behavior trees modularize robustness and safety in hybrid systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1482–1488. IEEE, 2014. (Cited on page 59.)
- [28] Michele Colledanchise and Petter Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics*, pages 372–389, 2016. (Cited on page 35.)
- [29] Michele Colledanchise and Petter Ögren. *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018. (Cited on pages 3, 20, 33, 36, 54, 58, 60, 62, and 68.)
- [30] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, pages 183–189, 2018. (Cited on pages 34 and 66.)

- [31] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022. (Cited on page 57.)
- [32] Michael T. Cox and Chen Zhang. Planning as mixed-initiative goal manipulation. In *Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling*, pages 282–291, 2005. (Cited on page 124.)
- [33] Stephen Cresswell and Peter Gregory. Generalised domain model acquisition from action traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 42–49, 2011. (Cited on page 82.)
- [34] Stephen Cresswell, Thomas L. McCluskey, and Margaret West. Acquisition of object-centred domain models from planning examples. *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 338–341, 2009. (Cited on page 82.)
- [35] Stephen Cresswell, Thomas L. McCluskey, and Margaret West. Acquiring planning domain models using LOCM. *The Knowledge Engineering Review*, page 195–213, 2013. (Cited on page 82.)
- [36] Pilar de la Cruz, Justus Piater, and Matteo Saveriano. Reconfigurable behavior trees: Towards an executive framework meeting high-level decision making and control layer features. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1915–1922, 2020. (Cited on page 64.)
- [37] Vadim Denisov, Dirk Fahland, and Wil M.P. van der Aalst. Repairing event logs with missing events to support performance analysis of systems with shared resources. In *Application and Theory of Petri Nets and Concurrency: 41st International Conference, PETRI NETS 2020, Paris, France, June 24–25, 2020, Proceedings 41*, pages 239–259. Springer, 2020. (Cited on page 133.)
- [38] David Cáceres Domínguez, Marco Iannotta, Johannes A. Stork, Erik Schaffernicht, and Todor Stoyanov. A stack-of-tasks approach combined with behavior trees: A new framework for robot control. *IEEE Robotics and Automation Letters*, pages 12110–12117, 2022. (Cited on pages 56, 59, 61, and 64.)
- [39] Eric Dortmans and Teade Punter. Behavior trees for smart robots practical guidelines for robot software development. *Journal of Robotics*, page 3314084, 2022. (Cited on pages 67 and 68.)
- [40] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017. (Cited on page 71.)

- [41] Stefan Edelkamp and Jörg Hoffmann. PDDL2.2: The language for the classical part of the 4th international planning competition. In *Proceedings of the ICAPS-2004 Workshop on the International Planning Competition*, pages 1–14, 2004. (Cited on page 29.)
- [42] Emanuel Eriksson. Design and development of a user interface for evaluating behavior tree interpretability in user studies. Master’s thesis, Örebro University, 2024. (Cited on page 69.)
- [43] Patrick Eyerich, Robert Mattmüller, and Georg Röger. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, pages 49–64. Springer, 2012. (Cited on page 107.)
- [44] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, pages 61–124, 2003. (Cited on pages 29 and 30.)
- [45] Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. In *Proceedings of IJCAI-17 Workshop on Explainable Planning*, 2017. (Cited on page 124.)
- [46] Khusniddin Fozilov, Yasuhisa Hasegawa, and Kosuke Sekiyama. Towards Self-Autonomy Evaluation using Behavior Trees. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 988–993, 2021. (Cited on page 64.)
- [47] Kevin French, Shiyu Wu, Tianyang Pan, Zheming Zhou, and Odest Chadwicke Jenkins. Learning behavior trees from demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7791–7797. IEEE, 2019. (Cited on pages 34, 35, 36, 40, 43, 63, and 69.)
- [48] Fabio Fusaro, Edoardo Lamon, Elena De Momi, and Arash Ajoudani. A Human-Aware Method to Plan Complex Cooperative and Autonomous Tasks using Behavior Trees. In *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, pages 522–529, 2021. (Cited on page 64.)
- [49] Chuanxiang Gao, Yu Zhai, Biao Wang, and Ben M. Chen. Synthesis and online re-planning framework for time-constrained behavior tree. In *2021 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1896–1901. IEEE, 2021. (Cited on page 64.)
- [50] Alfonso Gerevini and Derek Long. Plan constraints and preferences in PDDL 3: The language of the fifth international planning competition.

- Technical Report Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, 2005. (Cited on page 29.)
- [51] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004. (Cited on pages 24, 25, and 30.)
- [52] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016. (Cited on page 1.)
- [53] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Machine Learning Proceedings 1994*, pages 87–95. Elsevier, 1994. (Cited on page 82.)
- [54] Eleonora Giunchiglia, Michele Colledanchise, Lorenzo Natale, and Armando Tacchella. Conditional behavior trees: Definition, executability, and applications. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1899–1906, 2019. (Cited on page 23.)
- [55] Robert P. Goldman and Mark S. Boddy. Conditional linear planning. In *AIPS*, pages 80–85, 1994. (Cited on page 31.)
- [56] Martin C. Golumbic, Aviad Mintz, and Udi Rotics. Factoring and recognition of read-once functions using cographs and normality. In *Proceedings of the 38th annual Design Automation Conference*, pages 109–114, 2001. (Cited on page 36.)
- [57] Antonio González and Raúl Pérez. Improving the genetic algorithm of SLAVE. *Mathware & Soft Computing*, pages 59–70, 2009. (Cited on page 84.)
- [58] Alba Gragera and Alberto Pozanco. Exploring the limitations of using large language models to fix planning tasks. In *Proceedings of the ICAPS 2023 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2023. (Cited on page 124.)
- [59] Maxence Grand, Damien Pellier, and Humbert Fiorino. An accurate PDDL domain learning algorithm from partial and noisy observations. In *34th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 734–738. IEEE, 2022. (Cited on page 85.)
- [60] Peter Gregory and Alan Lindsay. Domain model acquisition in domains with action costs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 149–157, 2016. (Cited on page 82.)

- [61] Zhaoyuan Gu, Nathan Boyd, and Ye Zhao. Reactive locomotion decision-making and robust motion planning for real-time perturbation recovery. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1896–1902. IEEE, 2022. (Cited on pages 61 and 64.)
- [62] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*. Curran Associates Inc., 2023. (Cited on page 124.)
- [63] Simona Gugliermo. Learning planning domains for intelligent transport systems. In *Proceedings of the 35th Swedish Artificial Intelligence Society (SAIS'23) Annual Workshop*, 2023. (Cited on page 11.)
- [64] Simona Gugliermo, David Cáceres Domínguez, Marco Iannotta, Todor Stoyanov, and Erik Schaffernicht. Evaluating behavior trees. *Robotics and Autonomous Systems*, 2024. (Cited on page 11.)
- [65] Simona Gugliermo, Erik Schaffernicht, Christos Koniaris, and Federico Pecora. Learning behavior trees from planning experts using decision tree and logic factorization. *IEEE Robotics and Automation Letters*, pages 3534–3541, 2023. (Cited on pages 11, 57, 60, and 64.)
- [66] Simona Gugliermo, Erik Schaffernicht, Christos Koniaris, and Alessandro Saffiotti. Extracting planning domains from execution traces: a progress report. In *Proceedings of the ICAPS 2023 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 2023. (Cited on page 11.)
- [67] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011. (Cited on page 17.)
- [68] Blake Hannaford, Danying Hu, Dianmu Zhang, and Yangming Li. Simulation results on selector adaptation in behavior trees, 2016. (Cited on page 64.)
- [69] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, pages 191–246, 2006. (Cited on page 107.)
- [70] High-Level Expert Group on AI. Ethics Guidelines for Trustworthy AI, 2019. European Commission. (Cited on page 134.)
- [71] Jörg Hoffmann. FF: The Fast-Forward Planning System. *AI Magazine*, pages 57–57, 2001. (Cited on page 107.)

- [72] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. HDDL: An extension to PDDL for expressing hierarchical planning problems. In *Proceedings of the AAAI conference on artificial intelligence*, pages 9883–9891, 2020. (Cited on page 30.)
- [73] Richard Howey and Derek Long. VAL’s progress: The automatic validation tool for PDDL2.1 used in the international planning competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*. Citeseer, 2003. (Cited on pages 112 and 123.)
- [74] Qian Huang, Xianming Ma, Kun Liu, Xinyi Ma, and Weijian Pang. Autonomous reconnaissance action of swarm unmanned system driven by behavior tree. In *2022 IEEE International Conference on Unmanned Systems (ICUS)*, pages 1540–1544. IEEE, 2022. (Cited on pages 59 and 64.)
- [75] Yongfeng Huo, Jing Tang, Yinghui Pan, Yifeng Zeng, and Langcai Cao. Learning a planning domain model from natural language process manuals. *IEEE Access*, pages 143219–143232, 2020. (Cited on page 83.)
- [76] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, pages 141–154, 2011. (Cited on page 71.)
- [77] Zangir Iklassov and Dmitrii Medvedev. Robust reinforcement learning on graphs for logistics optimization. *arXiv preprint arXiv:2205.12888*, 2022. (Cited on page 84.)
- [78] Matteo Iovino, Julian Förster, Pietro Falco, Jen Jen Chung, Roland Siegwart, and Christian Smith. On the programming effort required to generate behavior trees and finite state machines for robotic applications. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5807–5813. IEEE, 2023. (Cited on pages 54, 57, 62, and 64.)
- [79] Matteo Iovino, Julian Förster, Pietro Falco, Jen Jen Chung, Roland Siegwart, and Christian Smith. Comparison between behavior trees and finite state machines. *arXiv preprint arXiv:2405.16137*, 2024. (Cited on page 54.)
- [80] Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. A survey of behavior trees in robotics and AI. *Robotics and Autonomous Systems*, 2022. (Cited on page 54.)

- [81] Matteo Iovino, Jonathan Styrud, Pietro Falco, and Christian Smith. Learning behavior trees with genetic programming in unpredictable environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4591–4597. IEEE, 2021. (Cited on pages 34 and 45.)
- [82] Seungwoo Jeong, Taekwon Ga, Inhwan Jeong, and Jongeun Choi. Behavior tree driven multi-mobile robots via data distribution service (dds). In *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, pages 1633–1638. IEEE, 2021. (Cited on pages 61 and 64.)
- [83] Rabia Jilani. Automated domain model learning tools for planning. In Mauro Vallati and Diane Kitchin, editors, *Knowledge Engineering Tools and Techniques for AI Planning*, pages 21–46. Springer, 2020. (Cited on page 2.)
- [84] Sergio Jiménez, Tomás De La Rosa, Susana Fernández, Fernando Fernández, and Daniel Borrajo. A review of machine learning for automated planning. *The Knowledge Engineering Review*, pages 433–467, 2012. (Cited on page 1.)
- [85] Simon Jones, Matthew Studley, Sabine Hauert, and Alan Winfield. Evolving behaviour trees for swarm robotics. In *Distributed autonomous robotic systems: The 13th international symposium*, pages 487–501. Springer, 2018. (Cited on page 63.)
- [86] Brendan Juba, Huyen Si Le, and Roni Stern. Safe learning of lifted action models. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 379–389, 2021. (Cited on page 83.)
- [87] Brendan Juba and Roni Stern. Learning probably approximately complete and safe action models for stochastic worlds. In *AAAI Conference on Artificial Intelligence*, 2022. (Cited on page 134.)
- [88] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, pages 99–134, 1998. (Cited on pages 30 and 134.)
- [89] Subbarao Kambhampati. Model-lite planning for the web age masses: the challenges of planning with incomplete and evolving domain models. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 1601–1604, 2007. (Cited on page 123.)
- [90] Valmeekam Karthik, Sarath Sreedharan, Sailik Sengupta, and Subbarao Kambhampati. Radar-x: An interactive interface pairing contrastive explanations with revised plan suggestions. *Proceedings of the AAAI Con-*

- ference on Artificial Intelligence*, pages 16051–16053, 2021. (Cited on page 134.)
- [91] Kei Kase, Chris Paxton, Hammad Mazhar, Tetsuya Ogata, and Dieter Fox. Transferable task execution from pixels through deep planning domain learning, 2020. (Cited on page 83.)
- [92] Joseph Kim, Christopher Banks, and Julie Shah. Collaborative planning with encoding of users’ high-level strategies. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 955–961, 2017. (Cited on page 124.)
- [93] George Kokotinis, George Michalos, Zoi Arkouli, and Sotiris Makris. A Behavior Trees-based architecture towards operation planning in hybrid manufacturing. *International Journal of Computer Integrated Manufacturing*, pages 324–349, 2024. (Cited on page 64.)
- [94] Jonas Kuckling, Antoine Ligot, Darko Bozhinoski, and Mauro Birattari. Behavior trees as a control architecture in the automatic modular design of robot swarms. In *International conference on swarm intelligence*, pages 30–43. Springer, 2018. (Cited on page 64.)
- [95] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013. (Cited on pages 18, 37, and 86.)
- [96] Max Kuhn and Ross Quinlan. *C50: C5.0 Decision Trees and Rule-Based Models*, 2022. R package version 0.1.6.9000. (Cited on page 18.)
- [97] Anagha Kulkarni, Siddharth Srivastava, and Subbarao Kambhampati. Planning for proactive assistance in environments with partial observability. *arXiv preprint arXiv:2105.00525*, 2021. (Cited on page 124.)
- [98] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, pages 239–286, 1995. (Cited on page 30.)
- [99] Leonardo Lamanna and Luciano Serafini. Action model learning from noisy traces: a probabilistic approach. *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 342–350, 2024. (Cited on pages 85, 103, and 107.)
- [100] Ning Li, Hao Jiang, Chunpeng Li, and Zhaoqi Wang. Towards adaptive behavior trees for robot task planning. In *2022 China Automation Congress (CAC)*, pages 6720–6725. IEEE, 2022. (Cited on pages 59 and 64.)
- [101] Johan Linde. Deep learning in automated planning for fleet management. Master’s thesis, Örebro University, 2024. (Cited on page 84.)

- [102] Alan Lindsay, Santiago Franco, Rubiya Reba, and Thomas L. McCluskey. Refining process descriptions from execution data in hybrid planning domain models. In *Proceedings of the International Conference on Automated Planning and Scheduling*, pages 469–477, 2020. (Cited on page [100](#).)
- [103] Ruikai Liu, Guangxi Wan, Maowei Jiang, Haojie Chen, and Peng Zeng. Autonomous robot task execution in flexible manufacturing: Integrating pddl and behavior trees in ariac 2023. *Biomimetics*, 2024. (Cited on page [122](#).)
- [104] Derek Long, Maria Fox, and Richard Howey. Planning domains and plans: Validation, verification and analysis. In *Proceedings of the ICAPS 2009 Workshop on Verification and Validation of Planning and Scheduling Systems (VVPS)*, 2009. (Cited on page [122](#).)
- [105] Artem Lykov and Dzmitry Tsetserukou. LLM-brain: AI-driven fast generation of robot behaviour tree based on large language model, 2024. (Cited on page [34](#).)
- [106] Yongjie Ma, Chuanshuai Deng, Jiexin Zhang, Yunlong Wu, Haoxiang Jin, and Yanzhen Wang. Resource scheduling in behavior trees. In *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2287–2292, 2022. (Cited on page [64](#).)
- [107] Zohar Manna. The correctness of programs. *Journal of Computer and System Sciences*, pages 119–127, 1969. (Cited on page [60](#).)
- [108] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. Towards a unified behavior trees framework for robot control. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 5420–5427. IEEE, 2014. (Cited on page [33](#).)
- [109] Matthias Mayr, Konstantinos Chatzilygeroudis, Faseeh Ahmad, Luigi Nardi, and Volker Krueger. Learning of parameters in behavior trees for movement skills. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7572–7579. IEEE, 2021. (Cited on page [34](#).)
- [110] Matthias Mayr, Konstantinos Chatzilygeroudis, Faseeh Ahmad, Luigi Nardi, and Volker Krueger. Learning of parameters in behavior trees for movement skills. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7572–7579. IEEE, 2021. (Cited on page [64](#).)
- [111] Thomas L. McCluskey, N. Elisabeth Richardson, and Ron M. Simpson. An interactive method for inducing operator descriptions. In *AIPS*, pages 121–130, 2002. (Cited on page [82](#).)

- [112] Thomas L. McCluskey, Tiago S. Vaquero, and Mauro Vallati. Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the 9th Knowledge Capture Conference*, pages 1–8, 2017. (Cited on page 100.)
- [113] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL-the Planning Domain Definition Language. *Technical Report CVC TR-98-003/DCS TR-1165*, 1998. (Cited on pages 3 and 29.)
- [114] Aviad Mintz and Martin Charles Golumbic. Factoring boolean functions using graph partitioning. *Discrete Applied Mathematics*, pages 131–153, 2005. (Cited on page 36.)
- [115] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. (Cited on page 16.)
- [116] Argaman Mordoch, Brendan Juba, and Roni Stern. Learning safe numeric action models. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12079–12086, 2023. (Cited on pages 84 and 103.)
- [117] Kira Mourão, Luke Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning strips operators from noisy and incomplete observations. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI’12, page 614–623, 2012. (Cited on pages 85, 103, and 108.)
- [118] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, pages 629–679, 1994. (Cited on page 86.)
- [119] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, pages 22071–22080, 2019. (Cited on page 62.)
- [120] Xenija Neufeld, Sanaz Mostaghim, and Sandy Brand. A hybrid approach to planning and execution in dynamic environments through hierarchical task networks and behavior trees. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 201–207, 2018. (Cited on pages 59, 64, and 122.)
- [121] Thu Nguyen, Alessandro Canossa, and Jichen Zhu. How human-centered explainable ai interface are designed and evaluated: A systematic survey, 2024. (Cited on page 134.)

- [122] Petter Ögren and Christopher I. Sprague. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, pages 81–107, 2022. (Cited on page 54.)
- [123] Conny Olz, Eva Wierzba, Pascal Bercher, and Felix Lindner. Towards improving the comprehension of htn planning domains by means of preconditions and effects of compound tasks. In *Proceedings of the 10th Workshop on Knowledge Engineering for Planning and Scheduling, KEPS*, 2021. (Cited on page 100.)
- [124] Ciprian Paduraru and Miruna Paduraru. Automatic difficulty management and testing in games using a framework based on behavior trees and genetic algorithms. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 170–179. IEEE, 2019. (Cited on pages 57 and 64.)
- [125] Ricardo Palma, Pedro A. González-Calero, Marco Antonio Gómez-Martín, and Pedro Pablo Gómez-Martín. Extending case-based planning with behavior trees. In *Proceedings of the 24th International Florida Artificial Intelligence Research Society Conference (FLAIRS-24)*, 2011. (Cited on page 34.)
- [126] Nilima Patil, Rekha Lathi, and Vidya Chitre. Comparison of c5.0 & cart classification algorithms using pruning technique. *International Journal of Engineering Research and Technology*, pages 1–5, 2012. (Cited on pages 35 and 37.)
- [127] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D Hager. Costar: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 564–571. IEEE, 2017. (Cited on pages 56 and 64.)
- [128] Chris Paxton, Felix Jonathan, Andrew Hundt, Bilge Mutlu, and Gregory D Hager. User experience of the costar system for instruction of collaborative robots, 2017. (Cited on page 63.)
- [129] Chris Paxton, Nathan Ratliff, Clemens Eppner, and Dieter Fox. Representing robot task plans as robust logical-dynamical systems. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5588–5595. IEEE, 2019. (Cited on page 64.)
- [130] Edwin P.D. Pednault. ADL: exploring the middle ground between STRIPS and the situation calculus. *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324—332, 1989. (Cited on page 31.)

- [131] Marco Pegoraro and Wil M.P. van der Aalst. Mining uncertain event data in process mining. In *2019 International Conference on Process Mining (ICPM)*, pages 89–96, 2019. (Cited on page 133.)
- [132] Renato de Pontes Pereira and Paulo Martins Engel. A framework for constrained and adaptive behavior-based agents, 2015. (Cited on pages 56 and 64.)
- [133] Geert Poels, Ann Maes, Frederik Gailly, and Roland Paemeleire. Measuring the perceived semantic quality of information models. In *International Conference on Conceptual Modeling*, pages 376–385. Springer, 2005. (Cited on page 100.)
- [134] Willard V. Quine. The problem of simplifying truth functions. *The American mathematical monthly*, pages 521–531, 1952. (Cited on page 36.)
- [135] J. Ross Quinlan. Induction of decision trees. *Machine learning*, pages 81–106, 1986. (Cited on page 35.)
- [136] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993. (Cited on page 18.)
- [137] Franco Raimondi, Charles Pecheur, and Guillaume Brat. PDVer: A Tool to Verify PDDL Planning Domains. In *Proceedings of the ICAPS, Workshop on Verification and Validation of Planning and Scheduling Systems*, 2009. (Cited on page 123.)
- [138] Alireza Rastegarpanah, Hector Cruz Gonzalez, and Rustam Stolkin. Semi-autonomous behaviour tree-based framework for sorting electric vehicle batteries components. *Robotics*, 2021. (Cited on page 64.)
- [139] Francisco Martín Rico, Matteo Morelli, Huascar Espinoza, Francisco J Rodríguez-Lera, and Vicente Matellán Olivera. Optimized Execution of PDDL Plans using Behavior Trees. In *AAMAS*, pages 1596–1598, 2021. (Cited on pages 64 and 121.)
- [140] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020. (Cited on page 85.)
- [141] Glen Robertson and Ian Watson. Building behavior trees from observations in real-time strategy games. In *2015 International symposium on innovations in intelligent systems and applications (INISTA)*, pages 1–7. IEEE, 2015. (Cited on pages 35, 57, and 64.)

- [142] Francesco Rovida, Bjarne Grossmann, and Volker Krüger. Extended behavior trees for quick definition of flexible robotic tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6793–6800. IEEE, 2017. (Cited on pages 55, 59, 64, and 85.)
- [143] Francesco Rovida, David Wuthier, Bjarne Grossmann, Matteo Fumagalli, and Volker Krüger. Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5964–5971. IEEE, 2018. (Cited on page 64.)
- [144] Stuart Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Viking, 2019. (Cited on page 135.)
- [145] Evgenii Safronov, Michael Vilzmann, Dzmitry Tsetserukou, and Konstantin Kondak. Asynchronous behavior trees with memory aimed at aerial vehicles with redundancy in flight controller. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3113–3118. IEEE, 2019. (Cited on page 64.)
- [146] Ismael Sagredo-Olivenza, Pedro Pablo Gómez-Martín, Marco Antonio Gómez-Martín, and Pedro Antonio González-Calero. Trained behavior trees: Programming by demonstration to support ai game designers. *IEEE Transactions on Games*, pages 5–14, 2017. (Cited on pages 34, 35, and 36.)
- [147] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, pages 353–362, 1983. (Cited on page 57.)
- [148] Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. *Unpublished ms. Australian National University*, page 27, 2010. (Cited on page 31.)
- [149] Emily Scheide, Graeme Best, and Geoffrey A. Hollinger. Synthesizing compact behavior trees for probabilistic robotics domains. *Autonomous Robots*, page 3, 2025. (Cited on page 36.)
- [150] Kirk Y.W. Scheper, Sjoerd Tijmons, Cornelis C. de Visser, and Guido C.H.E. de Croon. Behavior trees for evolutionary robotics, 2016. (Cited on pages 34, 64, and 66.)
- [151] José Á. Segura-Muros, Juan Fernández-Olivares, and Raúl Pérez. Learning numerical action models from noisy input data, 2021. (Cited on page 85.)

- [152] José Á. Segura-Muros, Raúl Pérez, and Juan Fernández-Olivares. Discovering relational and numerical expressions from plan traces for learning action models. *Applied Intelligence*, pages 7973–7989, 2021. (Cited on pages 84, 90, 103, and 108.)
- [153] Sailik Sengupta, Tathagata Chakraborti, Sarath Sreedharan, Satya Gautam Vadlamudi, and S. Kambhampati. Radar - a proactive decision support system for human-in-the-loop planning. In *AAAI Fall Symposia*, 2017. (Cited on page 124.)
- [154] Salihin Shoeeb and Thomas L. McCluskey. On comparing planning domain models. In *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2011)*, pages 8–9, 2011. (Cited on page 101.)
- [155] Sile Shu, Sarah Preum, Haydon M. Pitchford, Ronald D. Williams, John Stankovic, and Homa Alemzadeh. A behavior tree cognitive assistant system for emergency medical services. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6188–6195. IEEE, 2019. (Cited on page 63.)
- [156] Ron M. Simpson, Thomas L. McCluskey, Weihong Zhao, Ruth S. Aylett, and Christophe Doniat. GIPO: an integrated graphical tool to support knowledge engineering in AI planning. In *Proceedings of the 6th European Conference on Planning*, 2001. (Cited on page 82.)
- [157] Ho Chit Siu, Kevin Leahy, and Makai Mann. STL: Surprisingly Tricky Logic (for System Validation). In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8613–8620. IEEE, 2023. (Cited on page 63.)
- [158] Hendawan Soebhakti, Mochamad Rizal Fauzi, Tajdar Hal Ata, and Wanda Eka. Coach application for soccer robot. In *ICAE 2023: Proceedings of the 6th International Conference on Applied Engineering, ICAE 2023, 7 November 2023, Batam, Riau islands, Indonesia*, page 160. European Alliance for Innovation, 2024. (Cited on page 64.)
- [159] Ted Stanion and Carl Sechen. Boolean division and factorization using binary decision diagrams. *IEEE transactions on computer-aided design of integrated circuits and systems*, pages 1179–1184, 1994. (Cited on page 36.)
- [160] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. *arXiv preprint arXiv:1705.08961*, 2017. (Cited on page 83.)

- [161] Marvin Stuede, Timo Lerche, Martin Alexander Petersen, and Svenja Spindeldreier. Behavior-Tree-Based Person Search for Symbiotic Autonomous Mobile Robot Tasks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2414–2420, 2021. (Cited on page 64.)
- [162] Jonathan Styrud, Matteo Iovino, Mikael Norrlöf, Mårten Björkman, and Christian Smith. Automatic behavior tree expansion with llms for robotic manipulation, 2024. (Cited on page 34.)
- [163] Gavin Suddrey, Ben Talbot, and Frederic Maire. Learning and executing re-usable behaviour trees from natural language instruction. *IEEE Robotics and Automation Letters*, pages 10643–10650, 2022. (Cited on page 35.)
- [164] Tadewos G. Tadewos, Abdullah Al Redwan Newaz, and Ali Karimoddini. Specification-guided behavior tree synthesis and execution for coordination of autonomous systems. *Expert Systems with Applications*, 2022. (Cited on page 64.)
- [165] Pattaraporn Tulathum, Bunyapon Usawalertkamol, Gustavo Alfonso Garcia Ricardez, Jun Takamatsu, Tsukasa Ogasawara, and Kenichi Matsumoto. Human-robot interaction system for non-expert users in convenience stores using behavior trees. In *2022 IEEE/SICE International Symposium on System Integration (SII)*, pages 1072–1077. IEEE, 2022. (Cited on page 63.)
- [166] Mauro Vallati and Diane Kitchin. *Knowledge Engineering Tools and Techniques for AI Planning*. Springer, 2020. (Cited on page 1.)
- [167] Mauro Vallati and Thomas L. McCluskey. A quality framework for automated planning knowledge models. In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART)*, pages 635–644, 2021. (Cited on page 100.)
- [168] Tiago S. Vaquero, José R. Silva, and J. Christopher Beck. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the Knowledge Engineering for Planning and Scheduling (KEPS) Workshop, ICAPS 2011*, 2011. (Cited on page 123.)
- [169] Parikshit Verma, Mohammed Diab, and Jan Rosell. Automatic generation of behavior trees for the execution of robotic manipulation tasks. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4, 2021. (Cited on page 122.)

- [170] Albert Ren Rui Wang. *Algorithms for Multilevel Logic Optimization*. PhD thesis, University of California, Berkeley, 1989. (Cited on pages [13](#), [36](#), [37](#), [41](#), and [82](#).)
- [171] Adam Wathieu, Thomas R. Groechel, Haemin Jenny Lee, Chloe Kuo, and Maja J. Matarić. RE:BT-Espresso: Improving interpretability and expressivity of behavior trees learned from robot demonstrations. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 11518–11524. IEEE, 2022. (Cited on pages [36](#), [37](#), [38](#), [41](#), [44](#), [47](#), [48](#), [49](#), [57](#), [61](#), [62](#), and [64](#).)
- [172] Ruichao Wu, Sitar Kortik, and Christoph Hellmann Santos. Automated behavior tree error recovery framework for robotic systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6898–6904. IEEE, 2021. (Cited on pages [61](#) and [64](#).)
- [173] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, pages 364–381, 2022. (Cited on page [37](#).)
- [174] Yunlong Wu, Jinghua Li, Huadong Dai, Xiaodong Yi, Yanzhen Wang, and Xuejun Yang. micROS.BT: An Event-Driven Behavior Tree Framework for Swarm Robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9146–9153, 2021. (Cited on page [64](#).)
- [175] Zhanhao Xiao, Hai Wan, Hankui Hankz Zhuo, Jinxia Lin, and Yanan Liu. Representation learning for classical planning from partially observed traces, 2019. (Cited on page [84](#).)
- [176] Jiahua Xu, Yunhan Lin, Haotian Zhou, and Huasong Min. Generating manipulation sequences using reinforcement learning and behavior trees for peg-in-hole task. In *2022 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2715–2720. IEEE, 2022. (Cited on pages [61](#) and [64](#).)
- [177] Qiang Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, pages 12–24, 1990. (Cited on page [30](#).)
- [178] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, pages 107–143, 2007. (Cited on page [82](#).)
- [179] Qin Yang, Zhiwei Luo, Wenzhan Song, and Ramvijas Parasuraman. Self-reactive planning of multi-robots with dynamic task assignments. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 89–91. IEEE, 2019. (Cited on page [64](#).)

- [180] Shuo Yang, Xinjun Mao, Shuo Wang, and Yantao Bai. Extending Behavior Trees for Representing and Planning Robot Adjoint Actions in Partially Observable Environments. *Journal of Intelligent & Robotic Systems*, 2021. (Cited on page 64.)
- [181] Siqi Yi, Stewart Worrall, and Eduardo Nebot. A persistent and context-aware behavior tree framework for multi sensor localization in autonomous driving, 2021. (Cited on page 64.)
- [182] Josh Zapf, Marco Roveri, Francisco Martin, and Juan Carlos Manzanares. Constructing behavior trees from temporal plans for robotic applications, 2024. (Cited on page 121.)
- [183] Haotian Zhou, Huasong Min, and Yunhan Lin. An autonomous task algorithm based on behavior trees for robot. In *2019 2nd China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI)*, pages 64–70. IEEE, 2019. (Cited on page 64.)
- [184] Hankz Zhuo. Crowdsourced action-model acquisition for planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015. (Cited on page 82.)
- [185] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13*, page 2444–2450, 2013. (Cited on page 85.)
- [186] Hankz Hankui Zhuo, Jing Peng, and Subbarao Kambhampati. Learning action models from disordered and noisy plan traces. *arXiv preprint arXiv:1908.09800*, 2019. (Cited on page 85.)
- [187] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence*, pages 1540–1569, 2010. (Cited on page 82.)

PUBLICATIONS *in the series*
ÖREBRO STUDIES IN TECHNOLOGY

1. Bergsten, Pontus (2001) *Observers and Controllers for Takagi – Sugeno Fuzzy Systems*. Doctoral Dissertation.
2. Iliev, Boyko (2002) *Minimum-time Sliding Mode Control of Robot Manipulators*. Licentiate Thesis.
3. Spännar, Jan (2002) *Grey box modelling for temperature estimation*. Licentiate Thesis.
4. Persson, Martin (2002) *A simulation environment for visual servoing*. Licentiate Thesis.
5. Boustedt, Katarina (2002) *Flip Chip for High Volume and Low Cost – Materials and Production Technology*. Licentiate Thesis.
6. Biel, Lena (2002) *Modeling of Perceptual Systems – A Sensor Fusion Model with Active Perception*. Licentiate Thesis.
7. Otterskog, Magnus (2002) *Produktionstest av mobiltelefonantennerna i mod-växlande kammare*. Licentiate Thesis.
8. Tolt, Gustav (2003) *Fuzzy-Similarity-Based Low-level Image Processing*. Licentiate Thesis.
9. Loutfi, Amy (2003) *Communicating Perceptions: Grounding Symbols to Artificial Olfactory Signals*. Licentiate Thesis.
10. Iliev, Boyko (2004) *Minimum-time Sliding Mode Control of Robot Manipulators*. Doctoral Dissertation.
11. Pettersson, Ola (2004) *Model-Free Execution Monitoring in Behavior-Based Mobile Robotics*. Doctoral Dissertation.
12. Överstam, Henrik (2004) *The Interdependence of Plastic Behaviour and Final Properties of Steel Wire, Analysed by the Finite Element Method*. Doctoral Dissertation.
13. Jennergren, Lars (2004) *Flexible Assembly of Ready-to-eat Meals*. Licentiate Thesis.
14. Jun, Li (2004) *Towards Online Learning of Reactive Behaviors in Mobile Robotics*. Licentiate Thesis.
15. Lindquist, Malin (2004) *Electronic Tongue for Water Quality Assessment*. Licentiate Thesis.
16. Wasik, Zbigniew (2005) *A Behavior-Based Control System for Mobile Manipulation*. Doctoral Dissertation.

17. Berntsson, Tomas (2005) *Replacement of Lead Baths with Environment Friendly Alternative Heat Treatment Processes in Steel Wire Production*. Licentiate Thesis.
18. Tolt, Gustav (2005) *Fuzzy Similarity-based Image Processing*. Doctoral Dissertation.
19. Munkevik, Per (2005) "Artificial sensory evaluation – appearance-based analysis of ready meals". Licentiate Thesis.
20. Buschka, Pär (2005) *An Investigation of Hybrid Maps for Mobile Robots*. Doctoral Dissertation.
21. Loutfi, Amy (2006) *Odour Recognition using Electronic Noses in Robotic and Intelligent Systems*. Doctoral Dissertation.
22. Gillström, Peter (2006) *Alternatives to Pickling; Preparation of Carbon and Low Alloyed Steel Wire Rod*. Doctoral Dissertation.
23. Li, Jun (2006) *Learning Reactive Behaviors with Constructive Neural Networks in Mobile Robotics*. Doctoral Dissertation.
24. Otterskog, Magnus (2006) *Propagation Environment Modeling Using Scattered Field Chamber*. Doctoral Dissertation.
25. Lindquist, Malin (2007) *Electronic Tongue for Water Quality Assessment*. Doctoral Dissertation.
26. Cielniak, Grzegorz (2007) *People Tracking by Mobile Robots using Thermal and Colour Vision*. Doctoral Dissertation.
27. Boustedt, Katarina (2007) *Flip Chip for High Frequency Applications – Materials Aspects*. Doctoral Dissertation.
28. Soron, Mikael (2007) *Robot System for Flexible 3D Friction Stir Welding*. Doctoral Dissertation.
29. Larsson, Sören (2008) *An industrial robot as carrier of a laser profile scanner: Motion control, data capturing and path planning*. Doctoral Dissertation.
30. Persson, Martin (2008) *Semantic Mapping Using Virtual Sensors and Fusion of Aerial Images with Sensor Data from a Ground Vehicle*. Doctoral Dissertation.
31. Andreasson, Henrik (2008) *Local Visual Feature based Localisation and Mapping by Mobile Robots*. Doctoral Dissertation.
32. Bouguerra, Abdelbaki (2008) *Robust Execution of Robot Task-Plans: A Knowledge-based Approach*. Doctoral Dissertation.

33. Lundh, Robert (2009) *Robots that Help Each Other: Self-Configuration of Distributed Robot Systems*. Doctoral Dissertation.
34. Skoglund, Alexander (2009) *Programming by Demonstration of Robot Manipulators*. Doctoral Dissertation.
35. Ranjbar, Parivash (2009) *Sensing the Environment: Development of Monitoring Aids for Persons with Profound Deafness or Deafblindness*. Doctoral Dissertation.
36. Magnusson, Martin (2009) *The Three-Dimensional Normal-Distributions Transform – an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. Doctoral Dissertation.
37. Rahayem, Mohamed (2010) *Segmentation and fitting for Geometric Reverse Engineering. Processing data captured by a laser profile scanner mounted on an industrial robot*. Doctoral Dissertation.
38. Karlsson, Alexander (2010) *Evaluating Credal Set Theory as a Belief Framework in High-Level Information Fusion for Automated Decision-Making*. Doctoral Dissertation.
39. LeBlanc, Kevin (2010) *Cooperative Anchoring – Sharing Information About Objects in Multi-Robot Systems*. Doctoral Dissertation.
40. Johansson, Fredrik (2010) *Evaluating the Performance of TEWA Systems*. Doctoral Dissertation.
41. Trincavelli, Marco (2010) *Gas Discrimination for Mobile Robots*. Doctoral Dissertation.
42. Cirillo, Marcello (2010) *Planning in Inhabited Environments: Human-Aware Task Planning and Activity Recognition*. Doctoral Dissertation.
43. Nilsson, Maria (2010) *Capturing Semi-Automated Decision Making: The Methodology of CASADEMA*. Doctoral Dissertation.
44. Dahlbom, Anders (2011) *Petri nets for Situation Recognition*. Doctoral Dissertation.
45. Ahmed, Muhammad Rehan (2011) *Compliance Control of Robot Manipulator for Safe Physical Human Robot Interaction*. Doctoral Dissertation.
46. Riveiro, Maria (2011) *Visual Analytics for Maritime Anomaly Detection*. Doctoral Dissertation.

47. Rashid, Md. Jayedur (2011) *Extending a Networked Robot System to Include Humans, Tiny Devices, and Everyday Objects*. Doctoral Dissertation.
48. Zain-ul-Abdin (2011) *Programming of Coarse-Grained Reconfigurable Architectures*. Doctoral Dissertation.
49. Wang, Yan (2011) *A Domain-Specific Language for Protocol Stack Implementation in Embedded Systems*. Doctoral Dissertation.
50. Brax, Christoffer (2011) *Anomaly Detection in the Surveillance Domain*. Doctoral Dissertation.
51. Larsson, Johan (2011) *Unmanned Operation of Load-Haul-Dump Vehicles in Mining Environments*. Doctoral Dissertation.
52. Lidström, Kristoffer (2012) *Situation-Aware Vehicles: Supporting the Next Generation of Cooperative Traffic Systems*. Doctoral Dissertation.
53. Johansson, Daniel (2012) *Convergence in Mixed Reality-Virtuality Environments. Facilitating Natural User Behavior*. Doctoral Dissertation.
54. Stoyanov, Todor Dimitrov (2012) *Reliable Autonomous Navigation in Semi-Structured Environments using the Three-Dimensional Normal Distributions Transform (3D-NDT)*. Doctoral Dissertation.
55. Daoutis, Marios (2013) *Knowledge Based Perceptual Anchoring: Grounding percepts to concepts in cognitive robots*. Doctoral Dissertation.
56. Kristoffersson, Annica (2013) *Measuring the Quality of Interaction in Mobile Robotic Telepresence Systems using Presence, Spatial Formations and Sociometry*. Doctoral Dissertation.
57. Memedi, Mevludin (2014) *Mobile systems for monitoring Parkinson's disease*. Doctoral Dissertation.
58. König, Rikard (2014) *Enhancing Genetic Programming for Predictive Modeling*. Doctoral Dissertation.
59. Erlandsson, Tina (2014) *A Combat Survivability Model for Evaluating Air Mission Routes in Future Decision Support Systems*. Doctoral Dissertation.
60. Helldin, Tove (2014) *Transparency for Future Semi-Automated Systems. Effects of transparency on operator performance, workload and trust*. Doctoral Dissertation.

61. Krug, Robert (2014) *Optimization-based Robot Grasp Synthesis and Motion Control*. Doctoral Dissertation.
62. Reggente, Matteo (2014) *Statistical Gas Distribution Modelling for Mobile Robot Applications*. Doctoral Dissertation.
63. Långkvist, Martin (2014) *Modeling Time-Series with Deep Networks*. Doctoral Dissertation.
64. Hernández Bennetts, Víctor Manuel (2015) *Mobile Robots with In-Situ and Remote Sensors for Real World Gas Distribution Modelling*. Doctoral Dissertation.
65. Alirezaie, Marjan (2015) *Bridging the Semantic Gap between Sensor Data and Ontological Knowledge*. Doctoral Dissertation.
66. Pashami, Sepideh (2015) *Change Detection in Metal Oxide Gas Sensor Signals for Open Sampling Systems*. Doctoral Dissertation.
67. Lagriffoul, Fabien (2016) *Combining Task and Motion Planning*. Doctoral Dissertation.
68. Mosberger, Rafael (2016) *Vision-based Human Detection from Mobile Machinery in Industrial Environments*. Doctoral Dissertation.
69. Mansouri, Masoumeh (2016) *A Constraint-Based Approach for Hybrid Reasoning in Robotics*. Doctoral Dissertation.
70. Albitar, Houssam (2016) *Enabling a Robot for Underwater Surface Cleaning*. Doctoral Dissertation.
71. Mojtahedzadeh, Rasoul (2016) *Safe Robotic Manipulation to Extract Objects from Piles: From 3D Perception to Object Selection*. Doctoral Dissertation.
72. Köckemann, Uwe (2016) *Constraint-based Methods for Human-aware Planning*. Doctoral Dissertation.
73. Jansson, Anton (2016) *Only a Shadow. Industrial Computed Tomography Investigation, and Method Development, Concerning Complex Material Systems*. Licentiate Thesis.
74. Sebastian Hällgren (2017) *Some aspects on designing for metal Powder Bed Fusion*. Licentiate Thesis.
75. Junges, Robert (2017) *A Learning-driven Approach for Behavior Modeling in Agent-based Simulation*. Doctoral Dissertation.
76. Ricão Canelhas, Daniel (2017) *Truncated Signed Distance Fields Applied To Robotics*. Doctoral Dissertation.

77. Asadi, Sahar (2017) *Towards Dense Air Quality Monitoring: Time-Dependent Statistical Gas Distribution Modelling and Sensor Planning*. Doctoral Dissertation.
78. Banaee, Hadi (2018) *From Numerical Sensor Data to Semantic Representations: A Data-driven Approach for Generating Linguistic Descriptions*. Doctoral Dissertation.
79. Khaliq, Ali Abdul (2018) *From Ants to Service Robots: an Exploration in Stigmergy-Based Navigation Algorithms*. Doctoral Dissertation.
80. Kucner, Tomasz Piotr (2018) *Probabilistic Mapping of Spatial Motion Patterns for Mobile Robots*. Doctoral Dissertation.
81. Dandan, Kinan (2019) *Enabling Surface Cleaning Robot for Large Food Silo*. Doctoral Dissertation.
82. El Amine, Karim (2019) *Approaches to increased efficiency in cold drawing of steel wires*. Licentiate Thesis.
83. Persson, Andreas (2019) *Studies in Semantic Modeling of Real-World Objects using Perceptual Anchoring*. Doctoral Dissertation.
84. Jansson, Anton (2019) *More Than a Shadow. Computed Tomography Method Development and Applications Concerning Complex Material Systems*. Doctoral Dissertation.
85. Zekavat, Amir Reza (2019) *Application of X-ray Computed Tomography for Assessment of Additively Manufactured Products*. Doctoral Dissertation.
86. Mielle, Malcolm (2019) *Helping robots help us—Using prior information for localization, navigation, and human-robot interaction*. Doctoral Dissertation.
87. Grosinger, Jasmin (2019) *On Making Robots Proactive*. Doctoral Dissertation.
88. Arain, Muhammad Asif (2020) *Efficient Remote Gas Inspection with an Autonomous Mobile Robot*. Doctoral Dissertation.
89. Wiedemann, Thomas (2020) *Domain Knowledge Assisted Robotic Exploration and Source Localization*. Doctoral Dissertation.
90. Giaretta, Alberto (2021) *Securing the Internet of Things with Security-by-Contract*. Doctoral Dissertation.

91. Rudenko, Andrey (2021) *Context-aware Human Motion Prediction for Robots in Complex Dynamic Environments*. Doctoral Dissertation.
92. Eriksson, Daniel (2021) *Getting to grips with cartons: Interactions of cartonboard packages with an artificial finger*. Doctoral Dissertation.
93. Dinh-Cuong, Hoang (2021) *Vision-based Perception For Autonomous Robotic Manipulation*. Doctoral Dissertation.
94. Akalin, Nezih (2022) *Perceived Safety in Social Human-Robot Interaction*. Doctoral Dissertation.
95. Han Fan (2022) *Robot-aided Gas Sensing for Emergency Responses*. Doctoral Dissertation.
96. Tomic, Stevan (2022) *Human Norms for Robotic Minds*. Doctoral Dissertation.
97. Kondyli, Vasiliki (2023) *Behavioural Principles for the Design of Human-Centered Cognitive Technologies, The Case of Visuo-Locomotive Experience*. Doctoral Dissertation.
98. Morillo-Mendez, Lucas (2023) *SOCIAL ROBOTS / SOCIAL COGNITION. Robots' Gaze Effects in Older and Younger Adults*. Doctoral Dissertation.
99. Yang, Yuxuan (2023) *Advancing Modeling and Tracking of Deformable Linear Objects for Real-World Applications*. Doctoral Dissertation.
100. Adolfsson, Daniel (2023) *Robust large-scale mapping and localization*. Doctoral Dissertation.
101. Yang, Quantao (2023) *Robot Skill Acquisition through Prior-Conditioned Reinforcement Learning*. Doctoral Dissertation.
102. Landin, Cristina (2023) *AI-Based Methods For Improved Testing of Radio Base Stations: A Case Study Towards Intelligent Manufacturing*. Licentiate Thesis.
103. Larsson, Joakim (2024) *Have you heard about wire?, Monitoring of the wire drawing process*. Doctoral Dissertation.
104. Faridghasemnia, Mohamadreza (2024) *Grounding in Context: Studies in Robot Language Grounding in Real-world Contexts*. Doctoral Dissertation.

105. Pathi, Sai Krishna (2025) *AGIR: A Framework for Mobile Robots to Join Social Group Interactions*. Doctoral Dissertation.
106. Hazra, Rishi (2025) *Neurosymbolic Decision-Making with Large Language Models*. Doctoral Dissertation.
107. Tiago Rodrigues de Almeida (2025) *Learning to Understand and Predict Heterogeneous Trajectory Data*. Doctoral Dissertation.
108. Gugliermo, Simona (2025) *From Logs to Logic: Learning and Evaluating Interpretable Representations of Behavior for Autonomous Systems*. Doctoral Dissertation.