



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM – UNIVERSITY OF BOLOGNA

Department of Computer Science and Engineering

Master's Degree in Artificial Intelligence

Beyond LLMs: A Neuro-Symbolic Framework for Commonsense Reasoning in Robotics

Supervisor:
Prof. Maurizio Gabbrielli

Candidate:
Cono Cirone

Graduation Session II
Academic Year 2025/2026

Abstract

Domestic robots often receive underspecified instructions: where to place an object, which tool to use for a requested action, or whether their embodiment permits the task at all. Large language models (LLMs) can interpret varied language and draw on everyday associations, but their answers do not expose explicit premises. Symbolic systems make encoded knowledge and inference inspectable, but rely on representations and rules that may be incomplete. This thesis asks whether combining these approaches can preserve or improve LLM-only performance on embodied commonsense tasks while making the reasoning behind a decision easier to inspect.

The thesis evaluates two complementary designs on RoboCSKBench, a five-task embodied-commonsense benchmark. Pipeline 1 studies partially automated offline construction of Answer Set Programming rules for the Tidy Up and Tool Usage tasks through commonsense graphs, SOMA-inspired annotation, LLM rule proposals and Clingo verification. Pipeline 2 instead spans all five tasks: an LLM extracts controlled facts at runtime and SWI-Prolog applies knowledge-base evidence or hand-crafted rules. Overlap on two tasks permits comparison of rule-construction strategies, while Pipeline 2 provides breadth. Matched pure-LLM and pure-logic ablations use three Mistral model sizes, and Pipeline 2 is transferred unmodified to five out-of-distribution datasets.

The results indicate that symbolic reasoning helps only under particular conditions. It is most useful when the symbolic component can retrieve affordance or capability knowledge and apply it through constraints that match the task’s answer format; in these cases, explicit grounding can also reduce the gap between larger and smaller language models. When the task requires broad open-ended generation, or when an external dataset expects a decision structure that the symbolic selector was not designed for, the same architecture can become redundant or harmful. The thesis therefore supports a conditional view: neuro-symbolic reasoning can make decisions more traceable without necessarily reducing performance, but only when representation, knowledge and inference are aligned.

Contents

Abstract	i
1 Introduction	1
1.1 Context and Motivation	1
1.2 The Neural–Symbolic Trade-Off	1
1.3 Thesis Approach and Positioning	2
1.4 Research Questions and Objectives	3
1.5 Contributions and Thesis Structure	3
2 Background and Related Work	5
2.1 Embodied Commonsense in Robotics	5
2.2 Explicit Commonsense Representations	7
2.2.1 Commonsense Knowledge Graphs	7
2.2.2 Robot Ontologies	8
2.3 Large Language Models and Parametric Commonsense	9
2.3.1 Language Models for Embodied Action	9
2.4 Neuro-Symbolic Reasoning	10
2.4.1 ASP and Rule Induction	11
2.4.2 Prolog and SWI-Prolog	11
2.4.3 Event Calculus	12
2.5 Literature Synthesis	13
3 The RoboCSKBench Benchmark	14
3.1 Benchmark Design and Rationale	14
3.2 Task Descriptions	15
3.2.1 Tidy Up: Object Placement	15
3.2.2 Tool Usage: Affordance-Based Tool Selection	16
3.2.3 Meta-Reasoning: Hardware Capability	16
3.2.4 Table Setting: Cutlery and Plate Prediction	17
3.2.5 Procedural Knowledge: Temporal Step Ordering	17

CONTENTS

3.3	Data Sources and Construction	17
3.4	Evaluation Metrics	19
3.5	Connection to This Thesis	21
4	Neuro-Symbolic Rule Learning via LLM-Guided Induction	22
4.1	System Architecture	22
4.2	Knowledge Base Construction	23
4.2.1	CSKG Subgraph Extraction for Tidy-Up	23
4.2.2	CSKG Subgraph Extraction for Tool Usage	25
4.2.3	Semantic Object Filtering via LLM	26
4.2.4	SOMA Ontology Annotation via LLM	26
4.2.5	ASP Integrity Constraints for Validation	27
4.3	Inductive Logic Rule Learning	28
4.3.1	ILASP Example Generation	28
4.3.2	LLM-Guided Rule Induction with Clingo Verification	30
4.3.3	Rule Pruning and Minimalization	31
4.4	Online Inference Phase	32
4.4.1	Inference for Tidy-Up	33
4.4.2	Inference for Tool Usage	33
5	Runtime Neuro-Symbolic Reasoning with SWI-Prolog	35
5.1	Architecture Overview	35
5.1.1	Family A: KB-Grounded Neuro-Symbolic Loop	36
5.1.2	Family B: Direct LLM-to-Prolog Reasoning	37
5.2	Task-Specific Implementations	37
5.2.1	Tidy Up — Location KB and Prolog Location Selector	40
5.2.2	Tool Usage — Affordance KB and Prolog Tool Selector	41
5.2.3	Meta-Reasoning — Robot Capability Axioms in Prolog	42
5.2.4	Table Setting — Meal Property Extraction and Prolog Cutlery Rules	44
5.2.5	Procedural Knowledge — Temporal Property Extraction and Prolog Step Ordering	45
6	Experimental Evaluation	47
6.1	Experimental Setup	48
6.1.1	Models and Configurations	48
6.1.2	Ablation Study Design	48
6.1.3	Benchmark and Metrics	49
6.2	Results on RoboCSKBench	50

6.2.1	Tidy Up	50
6.2.2	Tool Usage	54
6.2.3	Meta-Reasoning	56
6.2.4	Table Setting	58
6.2.5	Procedural Knowledge	59
6.3	Cross-Task Analysis	61
6.3.1	Conditions for Symbolic Advantage	61
6.3.2	The Equalisation Effect	62
6.4	Discussion	63
6.4.1	Limitations of the Experimental Design	63
6.4.2	Design Principles for Neuro-Symbolic Systems	64
7	Generalization Study	66
7.1	Out-of-Distribution Datasets	67
7.1.1	VirtualHome	67
7.1.2	TidyBot	68
7.1.3	PIQA	69
7.1.4	BEHAVIOR-1K	69
7.1.5	RecipeNLG	70
7.2	Results	70
7.2.1	Tidy Up: VirtualHome and TidyBot	70
7.2.2	Tool Usage: PIQA	72
7.2.3	Meta-Reasoning: BEHAVIOR-1K	73
7.2.4	Procedural Knowledge: RecipeNLG	74
7.3	Cross-Dataset Analysis	75
7.4	Discussion	77
7.4.1	When and Why Symbolic Rules Generalise	77
7.4.2	Systemic Bottlenecks and Limitations	78
7.4.3	Implications for Hybrid Architectures	78
8	Conclusion	80
8.1	Summary of the Work	80
8.2	Answers to the Research Questions	80
8.3	Limitations	82
8.4	Future Work	83
8.5	Closing Remarks	84
	References	85

CONTENTS

A	Appendix	95
A.1	Extracted Domestic Subgraph Format	95
A.2	Extracted Tool Knowledge Base Format	96
A.3	Pipeline 1 Large Language Model Prompts	98
A.3.1	Semantic Object Filtering Prompt (Llama 3.1:8b)	98
A.3.2	SOMA Ontology Annotation Prompt (Llama 3.1:8b)	99
A.3.3	LLM-Guided Rule Induction System Prompt (Mistral Medium)	100
A.4	Pipeline 2 Large Language Model Prompts	101
A.4.1	Location Type Extraction Prompt (Tidy Up)	101
A.4.2	Affordance Extraction Prompt (Tool Usage)	103
A.4.3	Robot Capability Extraction Prompt (Meta-Reasoning)	104
A.4.4	Meal Property Extraction Prompt (Table Setting)	106
A.4.5	Recipe Step Property Extraction Prompt (Procedural Knowledge)	108
A.5	Pipeline 2 Prolog Programs	110
A.5.1	Tidy Up Reasoner	110
A.5.2	Tool Usage Reasoner	112
A.5.3	Meta-Reasoning Capability Matcher	113
A.5.4	Table Setting Cutlery and Plate Reasoner	117
A.5.5	Procedural Knowledge Temporal Reasoner	119

List of Tables

3.1	Overview of the five tasks in RoboCSKBench, adapted from [44].	15
3.2	Summary of the main data sources used to construct RoboCSKBench. . .	18
4.1	Statistics of object counts in the constructed knowledge bases before and after filtering.	27
4.2	Number of typed constants declared per task.	29
6.1	Tidy Up multi-choice results (383 questions). The full pipeline outperforms the pure-LLM baseline for all three model sizes, with the largest gain observed for Mistral Small (+16.7 pp).	51
6.2	Tidy Up open/generative results (383 objects). MRR is the primary metric. The pure-LLM baseline marginally outperforms the full pipeline across all model sizes.	52
6.3	Tool Usage multi-choice results (291 questions). The KB-grounded pipeline produces the largest absolute improvement observed in any task, with Mistral Small gaining +41.9 pp.	54
6.4	Meta-Reasoning binary classification results (9,576 questions). Pure Logic achieves the highest F1 (0.996), indicating near-perfect calibration of the hand-crafted Prolog axioms to the in-distribution benchmark.	56
6.5	Meta-Reasoning multi-choice results (4,788 questions). Pure Logic achieves the highest accuracy (0.943).	57
6.6	Table Setting results (100 meals). The full pipeline consistently outperforms both ablations on both metrics.	58
6.7	Procedural Knowledge binary results (3,104 questions). The full pipeline and pure-LLM baseline achieve comparable performance. Pure Logic fails near-randomly (≈ 0.48).	59
6.8	Procedural Knowledge multi-choice results (1,600 questions). The pure-LLM baseline marginally outperforms the full pipeline.	61

LIST OF TABLES

6.9	Summary of the neuro-symbolic advantage across all tasks (using Mistral Large, in-distribution). Δ measures the full pipeline’s gain over the pure-LLM baseline.	61
6.10	The equalisation effect: the full pipeline compresses the performance gap between model sizes. Δ_{LLM} is the pure-LLM gap between Mistral Large and Small; Δ_{Full} is the full-pipeline gap.	62
7.1	Summary of OOD datasets used in the generalisation study. All datasets are evaluated with the unmodified Pipeline 2 code. “Shift type” indicates the primary axis of distribution shift relative to RoboCSKBench.	67
7.2	Tidy Up OOD accuracy on VirtualHome (67 questions) and TidyBot (598 questions). Bold marks the highest accuracy within each dataset.	71
7.3	PIQA OOD results. The domain-filtered subset (710 Q) retains only household-related goals. The full pipeline underperforms the pure-LLM baseline in both subsets.	72
7.4	BEHAVIOR-1K OOD results (2,032 binary questions, derived from 250 sampled tasks and balanced positive/negative robot configurations). The full pipeline achieves the highest F1 across all model sizes, with the largest gain for Mistral Large (+29.2 pp F1).	73
7.5	RecipeNLG OOD results (2,000 binary questions, 500 recipes). The full pipeline outperforms the pure-LLM baseline for all model sizes, reversing the in-distribution pattern where pure LLM led marginally.	74
7.6	In-distribution vs. OOD performance comparison. Δ_{ID} is the full pipeline’s advantage over pure LLM on RoboCSKBench; Δ_{OOD} is the same on the OOD dataset. All deltas compare Mistral Large in both conditions to ensure a consistent baseline across rows. Note that different tasks use different primary metrics (Accuracy or F1), as indicated by the “Metric” column; cross-row magnitude comparisons should be made with this caveat in mind.	76

List of Figures

4.1	Architecture overview of Pipeline 1	24
4.2	Examples of learned, pruned ASP rules for the tidy-up task. Each rule derives a goes In/2 predicate from SOMA features.	32
5.1	Architecture of Family A (KB-Grounded Neuro-Symbolic Loop)	38
5.2	Architecture of Family B (Direct LLM-to-Prolog Reasoning)	39

Chapter 1

Introduction

1.1 Context and Motivation

Instructions given to a domestic robot are often incomplete. A command such as “tidy the room” does not specify where each object belongs and “set the table” leaves the required plate and cutlery unstated. People resolve these omissions through everyday knowledge, whereas a robot may recognise the instruction and possess the necessary manipulation skills while still lacking the background knowledge needed to infer what the instruction leaves unsaid. The practical problem considered in this thesis is how associations available through an LLM’s parametric knowledge can be combined with explicit knowledge and rules to supply these unstated assumptions without making the resulting decision impossible to inspect.

RoboCSKBench provides a controlled setting in which to study this problem across five household tasks [44]. Rather than treating the benchmark only as a test of how much an LLM knows, this thesis uses it to compare two ways of combining neural interpretation with explicit knowledge and rules. The thesis therefore investigates whether a hybrid neuro-symbolic system can achieve performance comparable to an LLM-only approach while providing a more traceable basis for its conclusions.

1.2 The Neural–Symbolic Trade-Off

LLMs offer flexible language interpretation and broad parametric associations, but their outputs do not expose dependable provenance and may contain unsupported facts [34, 33]. Symbolic systems make encoded knowledge and inference inspectable, yet they require representations and rules to be specified in advance; missing or incorrectly represented premises limit what their derivations can establish [3, 9].

Hybrid architectures divide these responsibilities in different ways. A learned model

may construct a symbolic representation for a separate reasoner, symbolic knowledge may constrain neural learning, or the components may exchange information repeatedly [3, 9]. Related systems combine language-model interpretation with feasibility scores or executable program interfaces [1, 23], while explicitly neuro-symbolic systems pair LLM-based extraction or planning with learned or hand-authored logic programs [7, 25]. These designs can add traceable constraints, but they also introduce an interface at which incorrect extraction, incomplete knowledge or mismatched rules can create new errors. The research problem is therefore empirical: whether the additional structure improves or preserves performance, and under which conditions it instead becomes redundant or harmful.

1.3 Thesis Approach and Positioning

This thesis investigates the neuro-symbolic design space through two pipelines that construct and apply symbolic knowledge in different ways. Although Pipeline 2 covers all five benchmark tasks, it does not make Pipeline 1 redundant because the pipelines isolate different methodological questions. Pipeline 1 tests whether reusable symbolic rules can be constructed with a partially automated offline process from external knowledge and examples. Pipeline 2 holds the hand-crafted rules fixed and tests whether neural semantic extraction can be separated from explicit runtime inference across a broader range of tasks. Their overlap on Tidy Up and Tool Usage is intentional: it allows the evaluation to compare alternative ways of constructing symbolic knowledge on the same problems. Pipeline 2 therefore addresses breadth, whereas Pipeline 1 addresses the rule-acquisition bottleneck that Pipeline 2 leaves in place.

Pipeline 1 addresses Tidy Up and Tool Usage through an offline rule-learning process. Commonsense graph knowledge is filtered and expressed with a SOMA-inspired vocabulary; an LLM then proposes Answer Set Programming (ASP) rules, which are tested against encoded examples using the Clingo reasoner. The method draws on Inductive Logic Programming (ILP), although it replaces direct ILASP search with LLM-guided hypothesis generation and symbolic verification. At runtime, the learned rules are applied to previously unseen benchmark objects.

Pipeline 2 covers all five RoboCSKBench tasks and performs reasoning at runtime. The LLM extracts task-relevant facts from natural language, after which SWI-Prolog applies either knowledge-base-grounded selection rules or task-specific constraints. These rules are hand-crafted rather than learned, so this pipeline does not remove the rule-authoring bottleneck typical of symbolic systems. Its role in the thesis is to provide the broader cross-task test of whether neural semantic extraction can be separated from ex-

1.4. RESEARCH QUESTIONS AND OBJECTIVES

PLICIT symbolic inference in a useful way.

The evaluation compares the complete systems with pure-LLM and pure-logic configurations. Pipeline 2 is tested with three Mistral model sizes, allowing the analysis to examine whether symbolic structure can compensate for lower neural capacity. Its prompts, rules and knowledge base are then transferred without task-specific modification to five out-of-distribution datasets. The aim is not to establish that symbolic reasoning is universally superior, but to identify when explicit knowledge and constraints help, when they become redundant or harmful, and how their contribution depends on representation and task structure.

1.4 Research Questions and Objectives

The thesis addresses four research questions:

- RQ1** Does a neuro-symbolic pipeline outperform a matched pure-LLM baseline, and how does this effect vary across tasks and answer formats?
- RQ2** To what extent does the symbolic layer compensate for differences in neural model capacity?
- RQ3** What do neural-only and symbolic-only ablations reveal about the contribution and limitations of each component?
- RQ4** To what extent does the runtime neuro-symbolic pipeline generalise to out-of-distribution data without task-specific adaptation?

Answering these questions requires more than comparing final scores. The experiments also examine knowledge-base coverage, semantic extraction, rule behaviour and answer format, since each can determine whether the symbolic component has useful information on which to operate. RQ1–RQ3 are addressed by the in-distribution evaluation in Chapter 6; RQ4 is addressed by the transfer study in Chapter 7.

1.5 Contributions and Thesis Structure

The thesis contributes two implemented neuro-symbolic pipelines and an evaluation of their behaviour under different task conditions. The first pipeline combines common-sense graph extraction, SOMA-inspired semantic annotation and LLM-guided ASP induction with Clingo verification. The second pipeline spans all five RoboCSKBench tasks by coupling LLM-based fact extraction with SWI-Prolog rules, and is evaluated through

matched ablations across three neural model sizes. The same runtime architecture is then tested on VirtualHome, TidyBot, PIQA, BEHAVIOR-1K and RecipeNLG, where changes in vocabulary, user preference, task structure and source data expose when the symbolic component remains useful and when it becomes a source of error. Together, these contributions support a task-dependent assessment of neuro-symbolic reasoning rather than a claim of general superiority.

Chapter 2 reviews embodied commonsense, explicit knowledge representations, LLM-based robotics and neuro-symbolic reasoning. Chapter 3 describes RoboCSKBench and its evaluation metrics. Chapters 4 and 5 present Pipeline 1 and Pipeline 2, respectively. Chapter 6 reports the in-distribution experiments and ablations, while Chapter 7 examines transfer to the five external datasets. Chapter 8 summarises the findings, limitations and directions for further work.

Chapter 2

Background and Related Work

Research on commonsense reasoning in cognitive robotics asks whether the flexibility of learned models can be combined with the inspectability of symbolic representations without sacrificing predictive performance. Large language models can recover many everyday associations and interpret varied expressions [34], yet their output alone does not reveal which knowledge supported a decision. By contrast, explicit resources and logic programs make encoded knowledge and the resulting derivation available for inspection, although this transparency does not ensure that the premises are correct or appropriate to the situation a robot currently perceives. Neuro-symbolic approaches bring these capabilities together by using learned models to interpret language and symbolic components to apply an explicit decision procedure [3, 9]. Their attraction lies not in an assumption of superior accuracy, but in the possibility of attaining performance comparable to LLM-only methods while making decisions easier to trace and audit. This trade-off frames the literature reviewed in this chapter.

2.1 Embodied Commonsense in Robotics

Commonsense knowledge comprises assumptions that people routinely leave unstated because they are taken for granted in everyday communication and action [43]. These assumptions become relevant to cognitive robotics when an agent encounters a situation that its controller does not describe exhaustively and must use background knowledge to interpret the available information. Töberg et al.’s systematic review finds substantial variation in how robotic systems acquire and use such knowledge, alongside a divide between formal knowledge representation research and cognitive robotics [43]. Obtaining a relevant fact is therefore only part of the problem: the fact must also be represented in a form that can affect the robot’s decision.

An appropriate action rarely follows from an isolated fact. Knowing the usual function

CHAPTER 2. BACKGROUND AND RELATED WORK

of an object is helpful only when that function is relevant to the activity taking place, and its relevance may change with the physical situation. The literature consequently treats commonsense as background knowledge through which an agent interprets its observations and possible actions, rather than as a collection of independent facts [43].

Embodied commonsense makes this relation to perception and action explicit. Whereas people acquire expectations about physical situations through everyday experience, a robot depends on the information made available by its architecture. This knowledge bridges an underspecified task and the physical actions through which it is carried out [43]. RoboCSKBench tests whether a system can recover such background knowledge for household situations [44].

Consider a robot instructed to wipe up a spilled drink. Perception may identify liquid on the floor and several nearby objects, while the instruction leaves the means of execution unspecified. Embodied commonsense connects these observations to action by allowing the robot to infer that the liquid should be removed and that an absorbent object, such as a cloth, would be suitable. This inference must also account for the robot’s body and surroundings: a cloth behind a closed door may be appropriate but inaccessible, while a heavy cleaning device may exceed the robot’s manipulation capabilities. This example illustrates why embodied commonsense involves more than language interpretation or factual recall; it relates perceived entities and possible actions to the current task and the agent’s physical situation [43].

Embodied AI benchmarks expose different points at which this background knowledge affects task execution. VirtualHome represents household activities as executable programs, making the structure of an activity explicit [35]. ALFRED instead requires an agent to derive an action sequence from an instruction and visual observations while accounting for changes produced by earlier actions [39]. BEHAVIOR-1K broadens the range of activities and physical phenomena, including interactions with liquids [22]. However, these benchmarks do not evaluate language understanding and commonsense reasoning as separate abilities. Understanding an instruction already requires an agent to use commonsense knowledge about the objects and actions it mentions. The benchmarks therefore assess whether the agent can interpret language and use that interpretation to select actions that are compatible with the environment.

Housekeep focuses more directly on the knowledge behind a household decision: it evaluates object placement against human judgements and then embeds that decision in a task requiring exploration and action [15]. TidyBot considers the additional problem of personalisation by using an LLM to infer placement preferences from examples supplied by one user [46]. The comparison separates two forms of knowledge that are easily conflated: a regularity across households can provide a useful default, yet it may still

2.2. *EXPLICIT COMMONSENSE REPRESENTATIONS*

conflict with the convention governing a particular home.

Taken together, these studies define embodied commonsense through its role in situated decisions. The agent needs knowledge that extends beyond previously encountered cases while remaining connected to the entities and actions represented by the robotic system. Commonsense knowledge graphs primarily provide broad background associations, whereas ontologies supply the domain structure through which those associations can be interpreted.

2.2 Explicit Commonsense Representations

Storing commonsense outside a learned model makes an assertion available for retrieval and inspection but it does not make that assertion immediately usable because the external resource and the robot may describe the same situation in different terms. Knowledge graphs address this mismatch by recording associations at scale, while ontologies approach it by specifying how the domain should be described.

2.2.1 Commonsense Knowledge Graphs

A knowledge graph represents concepts as nodes and uses labelled edges to state how they are related. ConceptNet 5.5 applies this representation to multilingual general knowledge [40]: *AtLocation*, for example, records where an object is commonly found, whereas *UsedFor* associates it with a function. The explicit edge makes the relation inspectable and, when source metadata is retained, provides provenance: a record of where the assertion originated. Provenance permits an assertion to be traced, but does not by itself establish that the assertion is correct or relevant to the current environment.

Although natural language concepts make ConceptNet easy to query, they also leave room for ambiguity because the same word may denote different concepts and an edge collected by one method may not carry the same evidence as an edge obtained in another way [40]. More importantly, the graph states a general association rather than a fact about the present environment; knowing where an object is usually stored neither shows where it is now nor establishes that the usual location matches the preference of the current user.

Commonsense resources are often developed independently and differ in both focus and modelling choices. CSKG consolidates seven such resources in a common graph, despite the sparse overlap between them [14]. Integration makes complementary assertions available through one representation, but it does not turn the component resources into a complete account of commonsense. A missing edge may simply reflect an omission in the source material, so graph absence cannot safely be interpreted as evidence that a rela-

tion is false. In practice, broader access must therefore be accompanied by task-specific selection and an explicit treatment of missing information.

Ascent++ pursues coverage through extraction from web content rather than through the integration of existing graphs, and its refinement process attempts to improve the assertions recovered from text [32]. Even after refinement, however, an assertion must be interpreted before a robot can use it. Taken together, these resources show that enlarging a graph does not separate coverage from reliability; instead, a larger set of assertions makes the selection of contextually appropriate knowledge more consequential.

2.2.2 Robot Ontologies

An ontology defines the concepts through which a domain is understood rather than merely recording that two expressions are associated. It can, for example, express the role that an object plays in an activity within a conceptual model shared by the components of a robotic system.

KnowRob demonstrates this approach by organising the knowledge of an autonomous personal robot in a representation that supports querying and inference [42]. Observations can therefore be interpreted through the model before they inform planning or control.

SOMA develops the representation of everyday activity by relating a physical occurrence to its social meaning, so that an object is described according to the role it plays in a particular situation [4]. The same object can consequently receive a different interpretation when the activity changes, while the ontology provides a common vocabulary in which information recovered from text or a graph can be expressed before a rule uses it.

SOMA's abstraction over physical and social context supports the transfer of knowledge between related situations, but using it with information from other sources creates an alignment problem [4]. Information obtained from language or another resource must be mapped to the ontology's vocabulary and the system must decide which role applies in the case at hand. The ontology makes the resulting description consistent with its conceptual model; it does not guarantee that the preceding mapping was correct.

Knowledge graphs and ontologies thus address different parts of the representation problem: the graph makes assertions available, while the ontology determines how they can be interpreted within the robotic domain. Their combination still leaves natural language knowledge to be mapped into the vocabulary expected by the reasoner.

2.3 Large Language Models and Parametric Commonsense

Knowledge can also be elicited from information encoded implicitly in a language model rather than retrieved as an explicit graph edge. Petroni et al. examined pretrained language models as implicit knowledge bases by presenting incomplete statements and testing whether the missing relational fact could be recovered without task-specific training [34]. Their results varied considerably across relations, providing evidence for recoverable factual associations without showing that retrieval is consistent or that a recovered fact applies to a robot’s present situation.

This parametric knowledge is difficult to audit because the generated answer does not expose a traceable source. The uneven recall observed by Petroni et al., together with the hallucination risks reported for GPT-4, shows why fluent output cannot be equated with reliable factual retrieval [34, 33]. This distinction matters in robotics because linguistic plausibility does not imply that an action is executable. An LLM may infer what an instruction probably requires but another component must determine whether the proposed action is compatible with the current environment and the robot’s available skills [1]. In this context, grounding means connecting the model’s linguistic proposal to what the robot can actually do in the situation it perceives. A related intention–action gap has also been observed in vision-language-action models, whose apparent understanding of an instruction does not necessarily transfer to successful execution under changed physical conditions [11].

2.3.1 Language Models for Embodied Action

Language-guided robotics addresses this grounding problem by placing model output inside an architecture that constrains how it can affect action. SayCan uses a language model to score the relevance of candidate actions and combines those scores with values associated with the robot’s learned skills [1]. The former component interprets the instruction, while the latter estimates whether the corresponding skill can succeed in the current state. The final choice consequently depends on both linguistic relevance and feasibility.

Code as Policies connects language to action through program generation, asking the model to produce code that calls functions exposed by the robot’s perception and control interfaces [23]. The generated program provides an executable structure but remains limited to the functions implemented by that interface. SayCan constrains a proposal through learned skills, whereas Code as Policies constrains it through the available program func-

tions; neither permits unrestricted model output to determine the robot’s behaviour.

That boundary matters more than the output format itself, since neither system turns an LLM into a symbolic reasoner or demonstrates that an external component prevents model error. Instead, the LLM interprets language before another part of the architecture determines what follows from the interpretation, making its role one of semantic mediation rather than independently reliable decision making. Neuro-symbolic approaches formalise this separation by assigning explicit inference to a symbolic component.

2.4 Neuro-Symbolic Reasoning

Surveys by Besold et al. and Garcez and Lamb describe neuro-symbolic AI as a broad spectrum of approaches that combine learned representations with explicit symbols and inference [3, 9]. At one end, symbolic knowledge can constrain a neural model during learning; at another, a neural model can construct the symbolic representation on which a separate reasoner operates. Intermediate designs may exchange information repeatedly or embed differentiable forms of logical structure. Neuro-symbolic AI therefore does not denote one fixed architecture, but a family of arrangements that assign different roles to learning, representation and inference.

The latter arrangement is relevant when language must be translated before rules can be applied. A neural model constructs a symbolic description, after which a reasoner determines what follows [3]. The model therefore performs semantic extraction rather than producing the final reasoning result, although the conclusion remains conditional on the translation and the symbolic vocabulary.

LLM2LAS demonstrates this pattern in story-based question answering. An LLM extracts semantic structures from text, Inductive Learning of Answer Set Programs (ILASP) learns ASP rules from them, and an ASP solver applies the resulting program to unseen questions [7]. Although the evaluation is not embodied, the study is relevant because the symbolic component is learned from examples rather than written entirely by a domain expert.

A recent preprint introduces CLMASP as a complementary approach to robotic planning. It uses an LLM to generate a skeletal plan and ASP to refine that plan with robot action knowledge, evaluating the combination in VirtualHome [25]. Unlike LLM2LAS, CLMASP does not learn the ASP rules; its contribution lies in using them to constrain an LLM-generated plan. The two systems therefore illustrate separate uses of symbolic reasoning after neural interpretation: learning a reusable rule program in one case and enforcing an existing action model in the other.

2.4. NEURO-SYMBOLIC REASONING

2.4.1 ASP and Rule Induction

Answer Set Programming (ASP) computes solutions from a problem represented by facts, rules and constraints. Its non-monotonic semantics allows a default conclusion to be withdrawn when an exception becomes available [8]. This makes ASP suitable for domains in which a general rule may admit explicit exceptions, but only after the task has been translated into predicates. ASP determines what follows from an encoding, not whether that encoding describes the physical situation correctly.

Clingo is an ASP reasoner that grounds and solves a logic program [12]. Here, *grounding* means replacing variables with constants before computing answer sets, not connecting language to perception and action. Clingo can test whether a proposed rule gives the expected conclusions for encoded examples, although the result provides evidence only within the supplied encoding.

Aker et al. applied ASP to collaborative housekeeping by embedding ConceptNet knowledge with geometric and temporal information in a high-level task representation [2]. Their focus is the use of semantic knowledge within an ASP model rather than rule induction. The application connects external commonsense to household reasoning while leaving the conversion from graph knowledge to task predicates as a separate design step.

Inductive Logic Programming (ILP) instead learns rules from examples and background knowledge within a specified hypothesis space [31]. ILASP applies this principle to ASP, producing an answer set program as its hypothesis. Its noise-tolerant formulation assigns penalties to uncovered examples so that occasional errors can be considered during selection [20]. The supplied predicates and admissible hypotheses nevertheless determine what the learner can express.

Meli et al. used ILASP to refine an incomplete domain model for autonomous surgical planning. From executions of a training task, the system learned previously unknown axioms governing the task [28]. The application demonstrates the induction of answer set rules in robotics, although its setting differs substantially from household commonsense.

These applications motivate the rule-learning setting used later in the thesis, while Chapter 4 explains why Pipeline 1 replaces direct ILASP search with LLM-proposed rules that are checked using Clingo.

2.4.2 Prolog and SWI-Prolog

Prolog follows a different execution model because it attempts to prove a query by searching through clauses whose conclusions may satisfy it, rather than computing stable models of the program as a whole. SWI-Prolog provides an implementation designed for integration into software applications [45]. The operational contrast is therefore between

evaluating a runtime query through goal-directed search and computing answer sets that satisfy a complete ASP encoding.

KnowRob provides an established robotics application in which Prolog-based reasoning allows an autonomous personal robot to query formal knowledge about everyday manipulation [42]. Prolog serves as a runtime reasoning interface rather than as a method for learning the rules themselves.

Saccon et al. connect this logic-programming tradition to generative models through a robotic framework whose Prolog knowledge base can be populated from natural-language text with LLM assistance and subsequently used for planning [36]. The work is a close architectural precedent for separating neural knowledge extraction from symbolic use, although its Prolog representation and planning objective differ from the predicate extraction and decision rules studied in this thesis.

Together, these systems motivate runtime architectures in which current information is asserted as facts before a query is evaluated. Chapter 5 instantiates this arrangement with LLM-extracted facts and hand-written SWI-Prolog rules rather than learning the rules from examples.

The transparency of a logic program applies to its derivation, not necessarily to its premises. A trace remains misleading if the LLM supplied the wrong predicate, so symbolic inference makes the encoded decision procedure inspectable without establishing that language was translated correctly.

2.4.3 Event Calculus

A closely related formalism in logic programming is the *Event Calculus* [18, 38], which explicitly reasons about actions, events, and their effects over time. While static rules evaluate conditions at a single point in time, the Event Calculus represents a timeline where events occur and cause properties of the world (called *fluents*) to start or stop holding. Because properties are assumed to persist until an action explicitly alters them, the formalism provides a simple and principled way to model change without having to re-state what remains unchanged (the classic frame problem [27]).

In robotics and embodied commonsense, this framework is especially relevant for *procedural knowledge* and future task planning. For instance, in multi-step activities such as cooking recipes or household manipulation, actions cannot be evaluated in isolation: slicing an ingredient changes its state and enables cooking it, while completing one step satisfies the preconditions for the next. In a neuro-symbolic setting, an LLM can interpret natural language instructions and propose candidate actions, while an Event Calculus program in Prolog or ASP tracks how the environment evolves and verifies that the sequence of steps is temporally and causally valid [21].

2.5 Literature Synthesis

The literature reveals complementary weaknesses rather than a hierarchy of representations. Knowledge graphs broaden access to explicit assertions but remain incomplete and context-dependent, while ontologies impose a shared conceptual structure only after external information has been aligned with their vocabulary. Language models can recover associations and interpret linguistic variation, yet their outputs lack dependable provenance and may not correspond to the situation perceived by a robot. Logic and action formalisms can then enforce explicit decision procedures or temporal projections, but they cannot retrospectively establish that their premises were extracted or grounded correctly.

The methodological implication is a staged relation in which information must first be recovered and interpreted before explicit rules determine what follows. Because each transition can introduce an error, the value of combining neural and symbolic components remains an empirical question rather than a general guarantee. The later method chapters instantiate this relation in two ways, and the evaluation on RoboCSKBench examines when the additional structure helps.

Chapter 3

The RoboCSKBench Benchmark

This chapter presents RoboCSKBench, which is used throughout this thesis to evaluate embodied commonsense reasoning for household robotics. It was proposed as a language-driven, multi-task benchmark for assessing whether large language models and knowledge-based systems can infer the background knowledge required by embodied agents in domestic environments [44]. RoboCSKBench defines the task families, input-output formats, and evaluation criteria used here to compare language-model baselines with the neuro-symbolic pipelines.

3.1 Benchmark Design and Rationale

Robots that act in household environments require more than perception, navigation, and low-level control. Many domestic tasks are open-ended: the goal is expressed in natural language, but the instruction does not explicitly contain all the knowledge needed for execution. A command such as “tidy up the room” does not say where each object belongs; “set the table for dinner” does not list the required plate and cutlery; and “use a tool to open this package” does not specify which object has the appropriate affordance. Humans solve these problems by relying on commonsense knowledge about everyday objects, their locations, their physical properties, and the temporal structure of actions.

RoboCSKBench targets this type of knowledge by focusing on *embodied commonsense*, understood as the information an agent needs to interact appropriately with a dynamic everyday environment [44]. As introduced in Chapter 1, this connects background knowledge to what an agent perceives and can do. The benchmark is positioned between two evaluation traditions. Natural-language commonsense benchmarks test general reasoning abilities but are often detached from robotic action, whereas manipulation benchmarks evaluate embodied execution while frequently assuming that the relevant commonsense knowledge is already encoded in the task setup. RoboCSKBench fills part of this

3.2. TASK DESCRIPTIONS

Table 3.1: Overview of the five tasks in RoboCSKBench, adapted from [44].

Task	Reasoning capability	Question type	Size
Tidy Up	Object–location commonsense	Generative ranking	383
Tool Usage	Affordance-based tool selection	Multiple choice	291
Meta-Reasoning	Robot capability assessment	Single choice / binary	9,576
Table Setting	Plate and cutlery prediction	Multiple choice / set prediction	200
Procedural Knowledge	Temporal step ordering	Single choice / binary	3,104

gap by asking whether a system can infer the background knowledge that a household robot would need before acting.

The benchmark is organised as a multi-task framework because embodied commonsense is not a homogeneous capability. Knowing where an object belongs differs from identifying a suitable tool, and neither judgement determines whether a robot can physically perform an action. RoboCSKBench therefore defines five tasks: Tidy Up, Tool Usage, Meta-Reasoning, Table Setting, and Procedural Knowledge. Each task isolates a different form of embodied commonsense within a natural-language evaluation problem. Table 3.1 summarises their reasoning capabilities, question formats and sizes.

3.2 Task Descriptions

3.2.1 Tidy Up: Object Placement

The Tidy Up task evaluates whether a system can infer where household objects are typically located or stored, a basic requirement for domestic robots since a robot cannot tidy a room unless it knows the expected destination of objects that are out of place. The task is therefore grounded in object–location commonsense. For example, a toothbrush is normally associated with a bathroom, a saucepan with a kitchen, and a blanket with a bedroom or closet.

The task is available in generative-ranking and multiple-choice formats. In the former, the model receives an object and returns a ranked list of plausible household locations. More than one answer may be acceptable: a mug, for example, may be associated with a kitchen, cupboard, sink, or dishwasher, so the evaluation rewards systems that place plausible locations near the top of the ranking. In the multiple-choice format, the system instead selects the most plausible location from a fixed candidate set. Open ranking

therefore measures generative breadth, whereas multiple choice measures discrimination among the alternatives provided.

3.2.2 Tool Usage: Affordance-Based Tool Selection

The Tool Usage task evaluates affordance reasoning. In robotics, an affordance describes what an agent can do with an object under a particular relationship between the object’s properties, the agent’s capabilities and the environment [37]. A knife affords cutting, a sponge affords absorbing, a hammer affords striking, and a container affords holding. Such knowledge is essential for flexible action because a robot often receives a goal rather than a fully specified tool choice. RoboCSKBench formulates this task as a multiple-choice problem in which the system receives a task description and a list of candidate objects. Only one candidate provides the affordance needed for successful execution, so the model must infer that affordance from the description before selecting the corresponding object. For instance, if the task requires wiping up liquid, the system should prefer an absorbent object such as a sponge or cloth rather than an object that is merely common in a kitchen.

3.2.3 Meta-Reasoning: Hardware Capability

The Meta-Reasoning task evaluates whether a system can decide if a robot with a given hardware configuration is capable of executing a task. This is a form of self-assessment: before acting, an autonomous robot should know whether its body is physically suitable for the intended action. A stationary robot may not be able to perform a task requiring navigation; a one-armed robot may fail at tasks requiring simultaneous bimanual manipulation; and a low-degree-of-freedom arm may be insufficient for precise manipulation. Each benchmark instance pairs a natural-language instruction with a robot configuration describing its mobility, number of arms, degrees of freedom, gripper type, and gripper rigidity [44]. The system must then decide whether that configuration is sufficient for the task. In the binary format, each question asks whether one specified configuration can execute the instruction, and the answer is positive only when all required hardware capabilities are satisfied. Negative instances pair the same task with a configuration that violates at least one requirement. In the single-choice format, several robot configurations are presented and the system must select the one that is sufficient for the task.

3.3. DATA SOURCES AND CONSTRUCTION

3.2.4 Table Setting: Cutlery and Plate Prediction

The Table Setting task evaluates whether a system can infer the appropriate plate and cutlery for a meal. This task combines physical, cultural, and functional commonsense. A soup is normally served in a bowl and eaten with a spoon; a steak requires a knife and fork; sushi may require chopsticks; and a sandwich may be eaten by hand. The correct answer depends on the physical form of the food, the cuisine, and table-setting conventions. The task contains two subproblems. Plate prediction requires the system to select one plate type, such as a dinner plate, dessert plate, bowl, or coupe plate. Cutlery prediction is instead set-valued because several items may be appropriate for the same meal. Unlike standard multiple choice, this formulation allows a system to be partially correct when it predicts only some of the required items.

3.2.5 Procedural Knowledge: Temporal Step Ordering

The Procedural Knowledge task evaluates temporal reasoning over action sequences. Household tasks often impose ordering constraints: in cooking, ingredients are usually prepared before heating, an oven is preheated before baking, and food is plated before serving. A robot that executes steps in the wrong order may fail even if it understands each individual action. RoboCSKBench focuses on recipes. Each instance contains a recipe title and two steps from that recipe and the system must determine their chronological order, answering whether one step occurs before or after the other. This tests whether the model understands the procedural role of each step, not just the surface wording.

3.3 Data Sources and Construction

RoboCSKBench draws on different data sources because its tasks require different forms of commonsense evidence. These sources range from knowledge graphs and robotic manipulation datasets to recipe corpora and human judgements [44]. Table 3.2 relates each task to its principal data sources and explains how those sources contribute to benchmark construction.

For Tidy Up, the benchmark collects household objects and their typical locations. CSKG and Ascent++ provide commonsense triples involving location relations, while WordNet is used to filter objects and locations to household-relevant categories [14, 32, 29]. Microsoft COCO [24] contributes image-derived object-room associations, while AI2Thor [17] and Housekeep [15] provide data already grounded in household simulation or tidy-up tasks. Since the sources differ in reliability, the benchmark assigns higher trust to curated task-specific resources and lower trust to automatically extracted knowledge-

Table 3.2: Summary of the main data sources used to construct RoboCSKBench.

Task	Main sources	Construction role
Tidy Up	CSKG, Ascent++, Microsoft COCO, AI2Thor, Housekeep	Extract object–location tuples and aggregate them using source trust values.
Tool Usage	CSKG, AI2Thor, COAT, Narrative Objects, RoCS, Visual Affordance Dataset	Extract object–affordance pairs and map affordance variants to canonical labels.
Meta-Reasoning	ALFRED, DROID, RH20T, VirtualHome	Extract task demonstrations and corresponding robot hardware configurations.
Table Setting	Recipe1M+ and Prolific	Collect human judgements for plate and cutlery choices.
Procedural Knowledge	Recipe1M+ and GPT-4o-assisted step selection	Generate balanced temporal ordering questions from recipe-step pairs.

graph tuples. Duplicate tuples are merged, their trust values are accumulated, and low-confidence tuples are filtered out. The final Tidy Up dataset contains 383 objects with at least one associated location [44].

For Tool Usage, the benchmark constructs object–affordance pairs. Some sources explicitly describe affordances, while others describe object properties that must be mapped to affordance labels. For instance, hollowness may support a *contain* affordance. The authors normalize extracted affordances into representative labels, reducing lexical variation such as *heat* and *heating source*. The final resource contains 557 unique objects, 3,222 object–affordance links, and 86 affordances that are used to create 291 task instances.

For Meta-Reasoning, the benchmark extracts task–hardware pairs from robotic manipulation benchmarks, including ALFRED [39], DROID [16], RH20T [10], and VirtualHome [35]. Each source provides evidence that a particular hardware configuration can execute a task. The raw extraction contains many closely related instructions, so the authors cluster task sentences using sentence embeddings and select representative tasks. For each cluster, they derive a minimal hardware configuration and generate negative examples by pairing the same task with an insufficient configuration. This produces a balanced set of 9,576 task–hardware combinations.

For Table Setting, the benchmark authors found no suitable textual dataset and therefore used crowdsourcing [44]. Recipes are sampled from Recipe1M+ [26], and Prolific participants are asked which cutlery items and plate type they would use. In total, 51 annotators provide these judgements. The authors aggregate plate labels by majority

3.4. EVALUATION METRICS

vote and include cutlery items that pass an annotation threshold. This is necessary since table-setting conventions are partly cultural and are not readily represented in the existing knowledge sources used by the benchmark.

The Procedural Knowledge data are also derived from Recipe1M+ recipes [26]. GPT-4o is used during construction to identify relevant step pairs, which are then manually reviewed to remove unclear cases. The final questions are generated using before/after templates and balanced across positive and negative answers. This final balancing operation is important because otherwise a model could exploit label frequency rather than reason about temporal order.

3.4 Evaluation Metrics

Because RoboCSKBench contains different task formats, it cannot be evaluated with a single metric. The benchmark therefore uses measures suited to classification, ranking, and set-valued prediction [44].

Accuracy is used when each instance has a single correct answer. For N examples, it is defined as:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\hat{y}_i = y_i), \quad (3.4.1)$$

where $\hat{y}_i$ is the predicted label, y_i is the ground-truth label, and $\mathbb{I}$ is an indicator function. Accuracy is easy to interpret, but by itself it does not distinguish false positives from false negatives and is too strict for set-valued outputs.

For the Tidy Up generative task, RoboCSKBench uses ranking metrics rather than simple exact-match accuracy. The model returns a ranked list of possible locations for an object, and the prediction is compared against the set of ground-truth locations extracted during dataset construction. The first metric is Mean Reciprocal Rank (MRR), defined as:

$$\text{MRR} = \frac{1}{N} \sum_{i=1}^N \frac{1}{\text{rank}_i}, \quad (3.4.2)$$

where rank_i is the position of the first correct location in the ranked list for example i . MRR is useful because it rewards systems that place at least one correct location near the top of the list. This is appropriate for object placement, where several locations may be plausible but the most typical ones should be ranked first.

However, MRR only considers the first correct answer and ignores the rest of the ranked list. For this reason, the benchmark also reports mean average precision and mean average recall at fixed cut-offs $k \in \{1, 3, 5\}$. Precision at k measures how many of the first

k predicted locations are correct:

$$P@k = \frac{|\hat{L}_{1:k} \cap L|}{k}, \quad (3.4.3)$$

where $\hat{L}_{1:k}$ is the set of the first k predicted locations and L is the set of ground-truth locations. Recall at k measures how many of the ground-truth locations have been recovered within the first k predictions:

$$R@k = \frac{|\hat{L}_{1:k} \cap L|}{|L|}. \quad (3.4.4)$$

Averaging these values over all examples gives $\text{MAP}@k$ and $\text{MAR}@k$. These metrics complement MRR: $\text{MAP}@k$ rewards precision in the top-ranked predictions, while $\text{MAR}@k$ rewards coverage of the possible correct locations. This distinction is important because a system may rank one correct location first but still miss many other valid locations.

For binary tasks, precision, recall, and F1 provide a more detailed view than accuracy. They are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad (3.4.5)$$

$$\text{Recall} = \frac{TP}{TP + FN}, \quad (3.4.6)$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (3.4.7)$$

These metrics are especially relevant for Meta-Reasoning. A false positive means the system claims that a robot can perform a task when it cannot, which may be unsafe. A false negative means the system rejects a feasible task, reducing autonomy. F1 summarizes the balance between these two error types, although the individual precision and recall values remain important for interpretation.

Jaccard similarity is used for the set-valued cutlery predictions in Table Setting. Given a predicted set $\hat{S}$ and a ground-truth set S , it is defined as:

$$J(\hat{S}, S) = \frac{|\hat{S} \cap S|}{|\hat{S} \cup S|}. \quad (3.4.8)$$

This metric gives partial credit when the system predicts some, but not all, correct cutlery items. It also penalizes over-generation: predicting every possible utensil increases the union and lowers the score.

3.5 Connection to This Thesis

RoboCSKBench supports the investigation in this thesis because its tasks expose two distinct roles for symbolic structure. Tidy Up and Tool Usage depend strongly on factual object knowledge, making them suitable for evaluating knowledge-base-grounded reasoning. Meta-Reasoning, Table Setting, and Procedural Knowledge instead require properties expressed in language to be mapped into a form on which explicit rules can operate. These task groups provide the evaluation settings for Pipeline 1 and Pipeline 2, respectively, without assuming that the same design will benefit every form of commonsense reasoning.

The benchmark also supplies the input-output formats and metrics used to compare the neuro-symbolic systems with their pure-LLM and pure-logic ablations. Its task taxonomy is retained in the out-of-distribution evaluation, allowing the later chapters to examine whether the observed behaviour persists when the vocabulary, conventions, or source datasets change. The implementation and evaluation chapters describe these uses in detail.

Chapter 4

Neuro-Symbolic Rule Learning via LLM-Guided Induction

This chapter introduces the first of two neuro-symbolic frameworks developed in this thesis, designed to address the Tidy-Up and Tool Usage tasks within the RoboCSK-Bench [44]. By bridging the broad coverage of commonsense knowledge graphs (CSKGs) with the semantic reasoning capabilities of large language models (LLMs), the proposed pipeline learns interpretable Answer Set Programming (ASP) rules during an offline phase, allowing them to be executed at runtime using the Clingo solver [12].

4.1 System Architecture

To achieve this, the architecture is partitioned into two phases (illustrated in Figure 4.1):

1. **Offline Phase:** A domain-specific knowledge base is constructed from raw CSKG subgraphs, which are subsequently filtered and annotated with SOMA ontology predicates using a local LLM; these annotations are then validated symbolically to induce a set of general ASP rules that can generalize to objects not seen during training. For example, during this phase the system might learn a rule such as “an object with a `CookwareRole` that affords `FoodPreparationTask` belongs in the kitchen” from training objects like *pot* and *frying pan*.
2. **Online Phase:** At runtime, the pipeline handles previously unseen test objects, i.e., objects that were not part of the offline training set, by mapping their features to the SOMA domain through a single LLM query, followed by deterministic symbolic inference in Clingo using the pre-learned rules. Continuing the example above, if the test object *wok* is encountered at runtime, the LLM annotates it with

4.2. KNOWLEDGE BASE CONSTRUCTION

CookwareRole and FoodPreparationTask, and the previously learned rule fires to derive `goesIn(wok, kitchen)`.

To optimize computational efficiency, the workload is distributed across two distinct model configurations:

- **Llama 3.1:8b**[13] (via Ollama, running locally) is used for semantic object filtering and SOMA ontology annotation.
- **Mistral Medium**[30] (via the Mistral API) is used for rule induction during the offline phase and online tie-breaking.

This model split optimizes computational cost since the lightweight, local Llama model is sufficient for binary classification and feature annotation, whereas the more capable Mistral Medium model handles the complex logical induction and syntax required to propose valid ASP rules.

4.2 Knowledge Base Construction

Constructing the offline knowledge base involves a four-stage pipeline: (1) extracting relevant subgraphs from public commonsense knowledge graphs, (2) filtering the resulting concepts to isolate the household domain, (3) annotating the surviving objects with SOMA ontology classes, and (4) validating these annotations symbolically using ASP integrity constraints.

4.2.1 CSKG Subgraph Extraction for Tidy-Up

For the Tidy-Up task, this process begins with the extraction of a domestic-environment subgraph from the ConceptNet partition of CSKG [40]. To establish a representative spatial hierarchy, we execute a seed-based breadth-first traversal starting from 35 canonical household room types (such as *kitchen*, *bathroom*, or *garage*), discovering objects and their candidate locations by traversing only the `/r/AtLocation` relation, which encodes spatial associations (e.g., *knife AtLocation kitchen*).

Starting from those seed rooms, we include in our domestic subgraph all objects reachable via one or two `AtLocation` hops. The two-hop strategy captures transitive spatial relationships; specifically, it captures objects located *inside* furniture (hop 2) that is itself located in a room (hop 1) (e.g., *fork* $\xrightarrow{\text{AtLocation}}$ *drawer* $\xrightarrow{\text{AtLocation}}$ *kitchen*).

In addition to spatial locations, we extract the relations `/r/UsedFor`, `/r/HasProperty`, and `/r/CapableOf` from ConceptNet for the discovered objects. For the Tidy-Up task, all

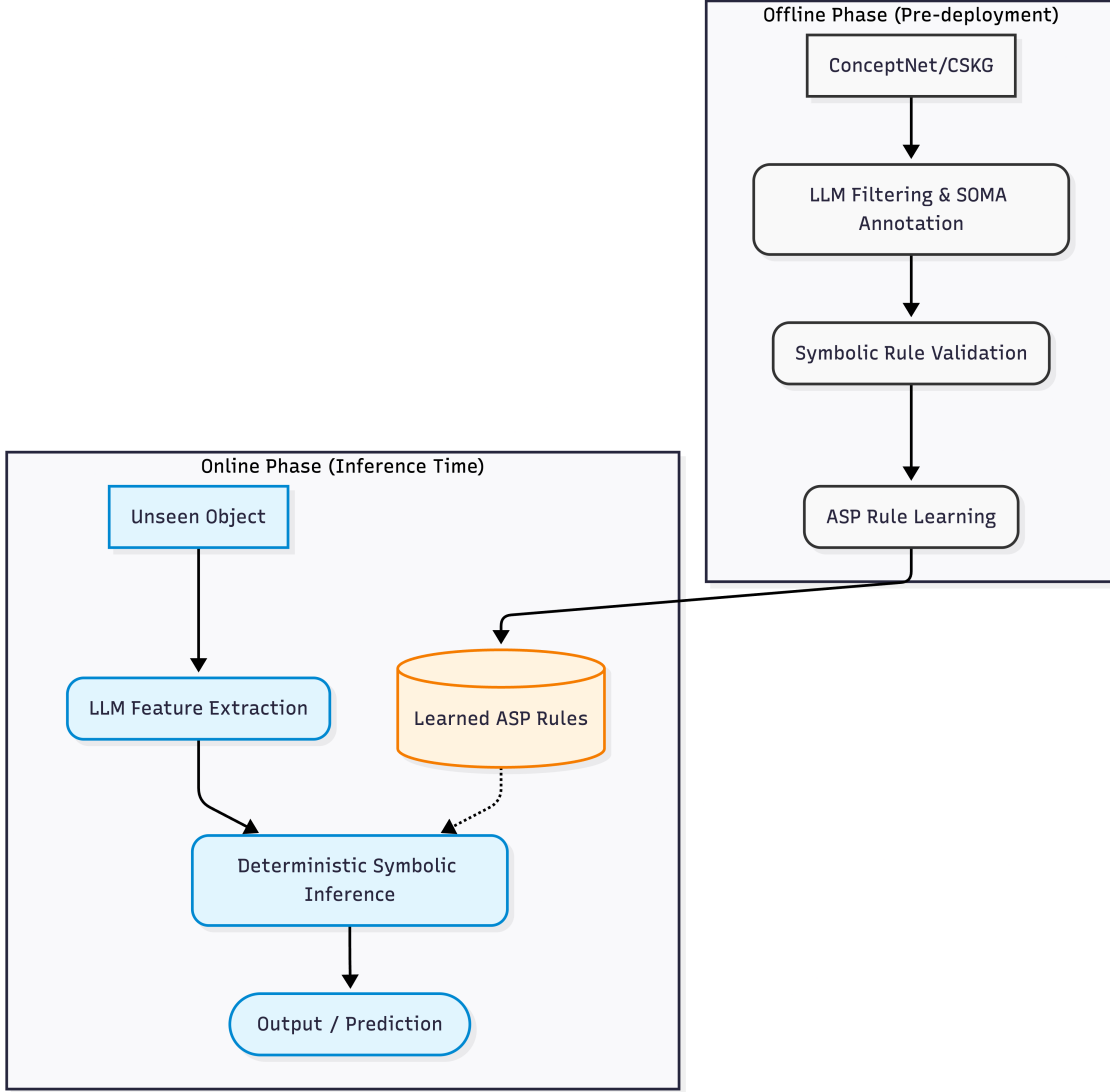


Figure 4.1: Architecture overview of Pipeline 1, demonstrating the interaction between the offline rule-learning phase and the online inference phase. The offline phase uses Llama 3.1:8b for semantic filtering and SOMA annotation, Mistral Medium for rule induction, and Clingo for symbolic verification; the online phase uses Mistral Medium for SOMA feature extraction and Clingo for rule-based inference. Blue nodes mark on-line inference, unfilled nodes mark offline construction, and the orange cylinder is the learned-rule store. Square-cornered boxes are inputs, rounded boxes are processing or output stages, and the frames separate the two phases; solid and dotted arrows show data flow and learned-rule transfer, respectively.

4.2. KNOWLEDGE BASE CONSTRUCTION

three relations are sourced exclusively from ConceptNet, because the task centres on spatial associations where ConceptNet provides the most reliable location-centric data; the noisier Visual Genome annotations are reserved for the Tool Usage task where affordance coverage is more critical. These additional relations are not used for graph discovery; instead, they are preserved as raw text descriptions to enrich the semantic context passed to the LLM during SOMA ontology annotation (Step 3).

Following extraction, URI labels are cleaned by removing articles ("a", "an", "the"), converting to lowercase, filtering for ASCII compatibility, and normalizing whitespace. The clean objects are stored in a JSON structure mapping them to their locations, source provenance, taxonomic history, and triggering seed rooms (see Appendix A.1 for an example).

4.2.2 CSKG Subgraph Extraction for Tool Usage

For the Tool Usage task, the knowledge base is enriched by consolidating data from multiple CSKG partitions [14], incorporating both ConceptNet (CN) and Visual Genome (VG) [19]. This extraction process operates via a three-phase pipeline tailored for tool-affordance discovery:

- **Phase 1 - Discovery.** The pipeline performs a breadth-first search starting from household room seeds via the `/r/AtLocation` relation in CSKG to discover candidate tools and their domestic contexts.
- **Phase 2 - Affordance Extraction.** For each discovered object, the system extracts tool-affordance properties using two specific relations: `/r/UsedFor` from ConceptNet (e.g., *broom UsedFor cleaning*) and `/r/CapableOf` from both ConceptNet and Visual Genome (e.g., *knife CapableOf cutting*). The resulting raw affordance phrases are then normalised to a canonical vocabulary through a three-tiered strategy:
 - **Direct matching:** If the extracted raw phrase matches exactly one of the 86 verbs in the predefined canonical affordance set, derived from the ground-truth affordance labels of the RoboCSKBench benchmark, it is accepted immediately (e.g., matching the raw phrase “*clean*” directly to the canonical label `clean`).
 - **Synonym and phrase mapping:** If the raw phrase is not in the canonical vocabulary, the system checks a manually curated mapping table that resolves common synonyms or action descriptions to canonical affordances (e.g., mapping “*drive in nails*” → `hammer`, or “*slice*” → `cut`). This table was con-

structured during development by analysing recurring synonym patterns in the extracted CSKG phrases.

- **First-verb fallback extraction:** For more complex phrases, the system extracts the first verb in the phrase (e.g., reducing “*cut cupboard*” to the verb *cut*) and checks whether it matches any canonical label.
- **Phase 3 — Inheritance.** For objects that lack extracted canonical affordances, the system performs a one-hop taxonomic traversal using */r/IsA* (restricted to ConceptNet sources) and inherits affordances from the parent concept. Two safeguards prevent noisy inheritance: (a) an explicit blocklist of overly generic parents (e.g., *tool*, *object*, *device*), and (b) a cardinality cap that skips parents yielding more than two distinct canonical affordances.

The resulting tool knowledge base maps each household object to its canonical affordances along with their source provenance (e.g., *cskg:cn*, *cskg:vg*, *cskg:cn:inherited*). An example of the generated JSON structure is provided in Appendix A.2.

4.2.3 Semantic Object Filtering via LLM

Because public commonsense graphs contain a high degree of noise, the raw extracted subgraphs initially include non-domestic or out-of-domain concepts, such as *airplane*, *whale*, or *casino*. To clean the dataset, we implement a semantic filtering phase using a local Llama 3.1:8b model to classify each object. This model is prompted with explicit guidelines defining household categories to accept (such as food, utensils, appliances, and tools) and out-of-domain categories to reject (such as non-domestic vehicles, wild animals, and abstract concepts), using a deterministic temperature of 0.0 (the complete system prompt is detailed in Appendix A.3.1):

- **Accepted categories examples:** food & drinks, utensils, appliances, clothing, toys, personal care, tools, stationery, medication, household items, decorations, musical instruments, safety equipment.
- **Rejected categories examples:** vehicles, weapons, wild animals, buildings, infrastructure, professions, abstract concepts, body parts, natural phenomena.

4.2.4 SOMA Ontology Annotation via LLM

To bridge the gap between unstructured commonsense text and structured, robot-interpretable reasoning, each validated object is subsequently annotated with semantic features from

4.2. KNOWLEDGE BASE CONSTRUCTION

Table 4.1: Statistics of object counts in the constructed knowledge bases before and after filtering.

Task / Knowledge Base	Initial Discovered	Filtered / Kept	Reduction (%)
Tidy-Up (ConceptNet)	1,346	645	52.08%
Tool Usage (CSKG)	2,823	1,561	44.70%

the SOMA (Socio-Physical Model of Activities) ontology [4]. This annotation phase assigns SOMA labels across three main dimensions:

1. `hasPhysicalQuality` — 23 labels capturing physical properties (e.g., `RequiresCooling`, `Perishable`, `Heated`, `Liquid`, `Granular`, `Heavy`, `Lightweight`, `Fragile`, `Rigid`).
2. `hasRole` — 17 labels capturing functional roles (e.g., `ConsumableRole`, `UtensilRole`, `CookwareRole`, `ToolRole`, `CleaningToolRole`, `ApplianceRole`, `StorageContainerRole`).
3. `affordsTask` — 12 labels capturing task affordances (e.g., `FoodPreparationTask`, `EatingDrinkingTask`, `MaintenanceTask`, `CleaningTask`, `HygieneTask`).

To maximize annotation accuracy, the LLM is guided by a few-shot prompt containing reference annotations for various object categories, complemented by an explicit error-correction section that lists common pitfalls encountered during development, as detailed in Appendix A.3.2. It should be noted that the reference annotations used in the few-shot examples may partially overlap with object categories present in the test set, which constitutes a minor limitation of the prompting strategy: in principle, the reference examples could provide indirect hints about the correct labels for some test objects.

4.2.5 ASP Integrity Constraints for Validation

Even with detailed prompting, the LLM occasionally produces annotations that contain logical contradictions. To address this, we apply a symbolic validation step using Clingo to enforce eight ASP integrity constraints, which automatically filter out or refine inconsistent object features:

1. **Toxicity–Consumability exclusion:** Toxic objects cannot have a `ConsumableRole` or food-related affordances, with an exception for medical supplies (perishable consumables that are neither cleaning nor maintenance items).
2. **Container–Liquid exclusion:** Storage containers cannot themselves be classified as liquids (e.g., a bottle has `HasContainer`, but is not `Liquid`).
3. **Fixture–Liquid exclusion:** Furniture and appliances cannot be liquids.

4. **Solid-Food–Granular exclusion:** Consumables that are both perishable and fragile (solid food items) cannot be granular.
5. **Non-cutting Utensil–Sharpness exclusion:** Eating utensils (e.g., forks, spoons) cannot be marked as sharp unless they also afford food preparation.
6. **Cooling–Heating exclusion:** Appliances marked as RequiresCooling or RequiresFreezing cannot simultaneously be Heated.
7. **Blade–Sharpness implication:** Any object annotated with HasBlade must also be annotated as Sharp.
8. **Tool-Specific Quality restriction:** IsRotary, HasAbrasiveSurface, and HasHeatElement are restricted to objects with a ToolRole, UtensilRole, or ApplianceRole.

4.3 Inductive Logic Rule Learning

Once a logically consistent knowledge base has been established, the pipeline proceeds to learn generalized ASP rules. While the methodology proposed is conceptually inspired by Inductive Logic Programming (ILP) frameworks such as ILASP [20], standard ILP solvers proved computationally intractable on our dataset; specifically, the high cardinality of SOMA features combined with the large volume of training examples caused traditional ILP systems to either exceed memory limits or time out. To circumvent these scalability bottlenecks, we designed a hybrid, LLM-guided rule induction loop that leverages Mistral Medium to propose candidate hypotheses and Clingo to verify them symbolically against the training examples.

4.3.1 ILASP Example Generation

Before the rule induction loop can be executed, the validated knowledge base must be compiled into an ILASP-compatible input format (`.las`) for both the Tidy-Up and Tool Usage tasks. The mode declarations define the search space for learned rules by specifying the acceptable head and body predicates. For the Tidy-Up task, the head mode declaration is:

- `#modeh(goesIn(var(obj), const(location)))` — maps an object variable to a location constant.

For the Tool Usage task, the head mode declaration is:

4.3. INDUCTIVE LOGIC RULE LEARNING

- `#modeh(hasAffordance(var(obj), const(affordance)))` — maps an object variable to a canonical affordance constant.

For both tasks, the rule bodies are defined using SOMA feature predicates:

- `#modeb(2, hasPhysicalQuality(var(obj), const(quality)))` — up to two physical quality conditions.
- `#modeb(2, hasRole(var(obj), const(role)))` — up to two role conditions.
- `#modeb(2, affordsTask(var(obj), const(task)))` — up to two task affordance conditions.

The `#maxv(1)` directive restricts the hypothesis space to single-variable rules (universal quantification over one object variable). This restriction is appropriate for the household domain because each benchmark question evaluates a single object independently; cross-object relationships (e.g., spatial co-occurrence of two objects) are not part of the task formulation, so multi-variable rules would only increase the search space without improving expressiveness.

Constant Declarations. All distinct values observed in the training knowledge base are declared as typed constants, as summarised in Table 4.2.

Table 4.2: Number of typed constants declared per task.

Task	Locations / Affordances	Physical Qualities	Roles	Task Affordances
Tidy-Up	24 locations	21	17	12
Tool Usage	86 affordances	21	17	12

Context-Dependent Training Examples. Within the ILASP formulation, training examples are modeled using a context-dependent structure where positive examples represent valid relationships occurring within the specific background knowledge context of an object’s SOMA features. For the Tidy-Up task, these positive examples map an object to its correct storage location:

```
#pos(eN@100, {goesIn(obj_name, location)}, {}, {
    obj(obj_name).
    hasPhysicalQuality(obj_name, q1).
    hasRole(obj_name, r1).
    affordsTask(obj_name, t1).
}).
```

In first-order logic (FOL) notation, this positive example expresses the following supervised assertion: given the background context that an object possesses specific SOMA features, the target relation should hold:

$$\text{obj}(o) \wedge \text{hasPhysicalQuality}(o, q_1) \wedge \text{hasRole}(o, r_1) \wedge \text{affordsTask}(o, t_1) \models \text{goesIn}(o, \ell)$$

A learned rule that covers this example would take the general form:

$$\forall x. \text{obj}(x) \wedge \text{hasPhysicalQuality}(x, q_1) \wedge \text{hasRole}(x, r_1) \implies \text{goesIn}(x, \ell)$$

For the Tool Usage task, the positive examples take a corresponding form:

```
#pos(eN@100, {hasAffordance(obj_name, affordance)}, {}, {
    obj(obj_name).
    hasPhysicalQuality(obj_name, q1).
    hasRole(obj_name, r1).
    affordsTask(obj_name, t1).
}).
```

Conversely, negative examples are generated implicitly under a closed-world assumption by identifying locations or affordances that are not associated with the object in the knowledge base. These are represented with an empty positive set and a non-empty negative set, and they are assigned a lower weight (e.g., 10) to account for the potential noise inherited from the initial CSKG graph extraction phase:

```
#pos(eN@10, {}, {goesIn(obj_name, incorrect_location)}, {
    obj(obj_name).
    ... SOMA BK context ...
}).
```

The generated .las file contains approximately 2,800 to 8,500 examples for the Tidy-Up task (depending on the negative sampling ratio), and approximately 2,011 examples for the Tool Usage task.

4.3.2 LLM-Guided Rule Induction with Clingo Verification

The core algorithmic contribution of this pipeline is the generate-and-test loop, which pairs Mistral Medium as a hypothesis generator with Clingo as a verification oracle (the complete system prompt is provided in Appendix A.3.3). Taking the Tidy-Up task as a concrete example, the pipeline executes the following iterative steps:

4.3. INDUCTIVE LOGIC RULE LEARNING

Step 1: Focus-Object Selection. To prevent the LLM from generating overly specific rules that fail to generalize, the system isolates the positive examples for a target location ℓ that are not yet covered by any previously accepted rule. From this uncovered set, a single *focus object* is selected, prioritizing the object annotated with the largest number of SOMA facts to provide a rich feature set, with ties broken alphabetically. Positive examples that repeatedly fail rule induction are temporarily deprioritized to prevent the generator from stalling on outlier cases.

Step 2: LLM Rule Proposal. Using the selected focus object as a concrete target, Mistral Medium is prompted to generate a candidate rule. To guide the generator, the prompt contains the ILASP mode declarations defining the search space, the focus object’s complete SOMA annotations (which prevents the LLM from hallucinating rules referencing non-existent feature combinations), the set of previously accepted rules to avoid redundancy, and the remaining uncovered positive examples to encourage generalization.

Step 3: Clingo Verification. Once a candidate rule r is proposed, it is evaluated by Clingo against the entire training set. During this step, Clingo computes two coverage metrics: the positive coverage (the set of true positives correctly derived by the rule) and the negative coverage (the set of false positives derived by the rule). Here, false positives are defined under the closed-world assumption: if a training object o is associated with ground-truth locations $\{l_1, l_2\}$, any derived relation $\text{goesIn}(o, l_3)$ where $l_3 \notin \{l_1, l_2\}$ counts as a violation. A proposed rule is accepted only if it successfully covers at least one new positive example without exceeding a configurable false-positive threshold (which is set to zero in the default configuration to enforce strict correctness).

Step 4: Iterative Refinement. This generate-and-test sequence repeats iteratively with newly selected focus objects. The loop terminates for a given target when all positive examples are successfully covered, when the LLM fails to propose an acceptable rule for a predefined number of consecutive iterations, or when a maximum iteration cap is reached. Once terminated, the induction process advances to the next location or affordance, ultimately accumulating the global rule set.

4.3.3 Rule Pruning and Minimalization

Following the induction phase, the raw rule set is processed through an algorithmic pruning phase to eliminate redundant body literals from the learned rules. This optimization runs as a greedy backward-elimination:

1. For each accepted rule r composed of a head h and a set of body literals $\{b_1, \dots, b_n\}$:

2. For each literal b_i tested sequentially:
 - (a) A candidate rule r' is constructed by removing b_i , yielding $r' = h \leftarrow \{b_1, \dots, b_{i-1}, b_{i+1}, \dots, b_n\}$.
 - (b) The candidate rule r' is evaluated against the training set via Clingo.
 - (c) If r' introduces no new negative coverage (indicating that the false-positive set remains unchanged or decreases) and the positives stay the same, the literal b_i is flagged as redundant and is permanently removed.
3. This cycle iterates until no further literals can be removed from any rule.

By removing non-essential conditions, this pruning phase yields minimal rules where every remaining body literal is strictly necessary to prevent false positives. This makes the final rule set not only more human-interpretable but also more robust against overfitting to co-occurring features in the training data. Examples of the resulting pruned rules are shown in Figure 4.2.

```
% --- kitchen ---
goesIn(X, kitchen) :- obj(X),
    hasPhysicalQuality(X, heated),
    hasRole(X, cookwareRole),
    affordsTask(X, foodPreparationTask).
goesIn(X, kitchen) :- obj(X),
    hasRole(X, consumableRole),
    affordsTask(X, eatingDrinkingTask).

% --- bedroom ---
goesIn(X, bedroom) :- obj(X),
    hasRole(X, beddingRole).
goesIn(X, bedroom) :- obj(X),
    affordsTask(X, sleepingTask).
```

Figure 4.2: Examples of learned, pruned ASP rules for the tidy-up task. Each rule derives a `goesIn/2` predicate from SOMA features.

4.4 Online Inference Phase

During the online execution phase, the pipeline predicts storage locations or tool choices for previously unseen objects from the RoboCSKBench. Although both tasks rely on SOMA feature mapping, they employ distinct reasoning strategies at runtime.

4.4. ONLINE INFERENCE PHASE

4.4.1 Inference for Tidy-Up

For the Tidy-Up task, inference proceeds sequentially:

1. **SOMA Feature Extraction:** The online phase begins by querying the LLM (Mistral Medium) to annotate the unseen object with SOMA classes, using the identical prompt schema employed during the offline training phase (Appendix A.3.2) to guarantee feature space consistency.
2. **Fact Generation:** The SOMA annotation is translated directly into ASP facts (e.g., `hasRole(hammer, toolRole).`).
3. **Clingo Inference:** Clingo evaluates these facts against the offline-induced rule set, deriving a set of candidate locations.
4. **Specificity-Weighted Ranking:** If Clingo derives multiple candidate locations, they are ranked based on the specificity of the rules that fired. The specificity score is determined by the number of body literals in each active rule, under the assumption that rules with more conditions represent stronger, more specific evidence.
5. **Tie-Breaking and Fallback Re-Ranking:** In the event of tied specificity scores, the LLM is invoked a second time to re-rank the tied locations based on semantic plausibility. Alternatively, if Clingo fails to derive any location because no rule fires, the system falls back to a direct, zero-shot LLM prediction. In practice, a diagnostic analysis on the Tidy-Up test set shows that Clingo provides location hints in approximately 78% of cases, while the remaining 22% require the LLM-only fallback path.

4.4.2 Inference for Tool Usage

Conversely, the Tool Usage task employs a task-driven selection strategy:

1. **Affordance Extraction:** The system extracts the required affordance from the user’s natural language task description using zero-shot Chain-of-Thought (CoT) reasoning.
2. **SOMA Feature Extraction & Symbolic Matching:** The candidate tools’ SOMA features are annotated using Mistral Medium (following the prompt in Appendix A.3.2) and matched symbolically against the learned `hasAffordance` rules in Clingo.

3. **Synonym Expansion:** To handle lexical variation in the user’s task descriptions, symbolic matching is augmented by a synonym expansion step that groups semantically equivalent verbs (such as grouping `cut`, `chop`, and `dice` into a single equivalence class).
4. **Tie-Breaking & Pre-Filtering Fallback:** If multiple tools match the required affordance symbolically, the LLM is used to break the tie. In contrast, if no tool matches, the system applies a SOMA-hint pre-filter to narrow the choices before falling back to direct LLM selection.

Chapter 5

Runtime Neuro-Symbolic Reasoning with SWI-Prolog

This chapter presents the second of the two pipelines developed in this thesis. Unlike Pipeline 1, which constructs symbolic rules through an offline induction loop (see Chapter 4), Pipeline 2 adopts a fundamentally different paradigm: rather than learning persistent rules from data, it performs *runtime neuro-symbolic reasoning* by coupling Mistral Medium (accessed via the Mistral API) with the SWI-Prolog logic engine at inference time. Under this design, the LLM is responsible for transforming unstructured natural-language input into structured symbolic facts, while Prolog applies hand-crafted domain rules to those facts to derive a decision. The approach eliminates the need for a training phase entirely and, crucially, allows the same pipeline to be applied to all five tasks of the RoboCSKBench [44], *Tidy Up*, *Tool Usage*, *Meta-Reasoning*, *Table Setting*, and *Procedural Knowledge*, without modifying the neural component or fine-tuning any model. As established in Chapter 1, the two pipelines are retained because broader task coverage does not answer Pipeline 1’s separate question about partially automated rule acquisition; their comparative trade-offs are revisited in Section 6.4.

5.1 Architecture Overview

The central design principle of Pipeline 2 is the decomposition of commonsense reasoning into two complementary competences: the neural competence of the LLM, which excels at semantic interpretation of natural language, and the symbolic competence of Prolog, which provides deterministic rule application. At inference time, the LLM produces a set of typed boolean or categorical facts, and Prolog consumes those facts to derive a decision.

The implementation of this principle differs substantially across the five benchmark

tasks, giving rise to two distinct *architectural families*. The distinction is not merely one of implementation detail but it reflects a fundamental difference in the locus of the final decision and in whether the pipeline requires access to an external knowledge base.

5.1.1 Family A: KB-Grounded Neuro-Symbolic Loop

The first architectural family governs the *Tidy Up* and *Tool Usage* tasks. These tasks share the property that the correct answer depends not only on the semantic interpretation of the query but also on factual knowledge about specific household objects that cannot be derived from the task description alone. In fact, determining that a *hammer* is the correct tool for a task requiring the hammer affordance, for instance, presupposes knowing that hammers possess that affordance, knowledge that must be retrieved from an object-affordance knowledge base rather than inferred from context.

While Family A addresses the same two tasks as Pipeline 1 (Chapter 4), the two differ fundamentally in their symbolic strategy. Pipeline 1 learns ASP rules offline from data via an LLM-guided induction loop and applies them at runtime using Clingo; Family A, by contrast, uses hand-crafted Prolog rules with no offline training phase and performs all reasoning at inference time by coupling an LLM extraction step with a Prolog candidate-selection step. This makes Family A simpler to deploy and extend to new tasks, whereas Pipeline 1 offers automated rule acquisition at the cost of annotation sensitivity.

To address the need for external knowledge, Family A implements a loop with up to three phases, as illustrated in Figure 5.1:

1. **Neural Phase 1 (Semantic Extraction).** The LLM analyses the natural-language query and extracts a structured semantic descriptor. For *Tidy Up*, this descriptor is the canonical location type to which the queried object belongs (e.g., kitchen, bathroom); for *Tool Usage*, it is the canonical physical affordance required by the task description (e.g., cut, hammer).
2. **Symbolic Phase (Prolog Candidate Selection).** The extracted descriptor is matched against a task-specific knowledge base (described in Section 4.2). For each candidate answer provided by the benchmark question, the Python driver program asserts Prolog facts encoding the candidate’s name (e.g., `candidate(kitchen)`), its knowledge-base-derived properties (e.g., `has_location/2` or `has_affordance/2`), and whether those properties are confirmed by the underlying commonsense knowledge graphs — ConceptNet for *Tidy Up*, or ConceptNet and Visual Genome for *Tool Usage* — via the `kb_confirmed/2` or `cn_confirmed/2` facts.
3. **Neural Phase 2 (Grounded Disambiguation).** If the symbolic phase cannot uniquely resolve an answer (i.e., it returns ambiguous or none), the LLM is re-invoked with

5.2. TASK-SPECIFIC IMPLEMENTATIONS

knowledge-base exemplars injected into the prompt to provide grounding context for the final selection. This third phase is task-specific and is described in detail in Sections 5.2.1 and 5.2.2.

The key property of this architecture is that the knowledge base acts as a bridge between the two neural phases: the LLM’s initial extraction identifies what kind of knowledge is needed, the KB supplies it, and the Prolog engine determines whether that evidence is sufficient for a deterministic answer or whether neural disambiguation is required.

5.1.2 Family B: Direct LLM-to-Prolog Reasoning

The second architectural family governs the *Meta-Reasoning*, *Table Setting*, and *Procedural Knowledge* tasks. Unlike Family A, these tasks do not query a local, structured database file at runtime. Instead, the final decision is derived from semantic properties extracted directly from the query. For *Meta-Reasoning* and *Procedural Knowledge*, these properties are specified or implied directly in the input text (e.g., kinematic capabilities or recipe steps). For *Table Setting*, however, properties such as whether a dish is “Asian cuisine” or “needs cutting” sit on the boundary: while the pipeline does not query an external database file, it relies on the LLM to map its latent parametric world knowledge into a strict symbolic domain.

Family B therefore implements a simpler two-phase pipeline:

1. **Neural Phase (Property Extraction).** The LLM analyses the input and produces a structured JSON dictionary of boolean or categorical semantic properties. The property schema is defined in the system prompt, which also provides few-shot examples with chain-of-thought reasoning to calibrate extraction accuracy.
2. **Symbolic Phase (Prolog Rule Application).** The extracted properties are dynamically asserted as Prolog facts and passed to a hand-crafted Prolog program, which derives the final answer deterministically. Crucially, the Prolog output *is* the final answer since there is no second LLM invocation. Automating the synthesis of these task-specific Prolog programs, for instance through program induction from examples, constitutes an interesting direction for future work.

5.2 Task-Specific Implementations

The following sections describe how the two architectural families are instantiated for each of the five benchmark tasks. Each description follows a consistent structure: the

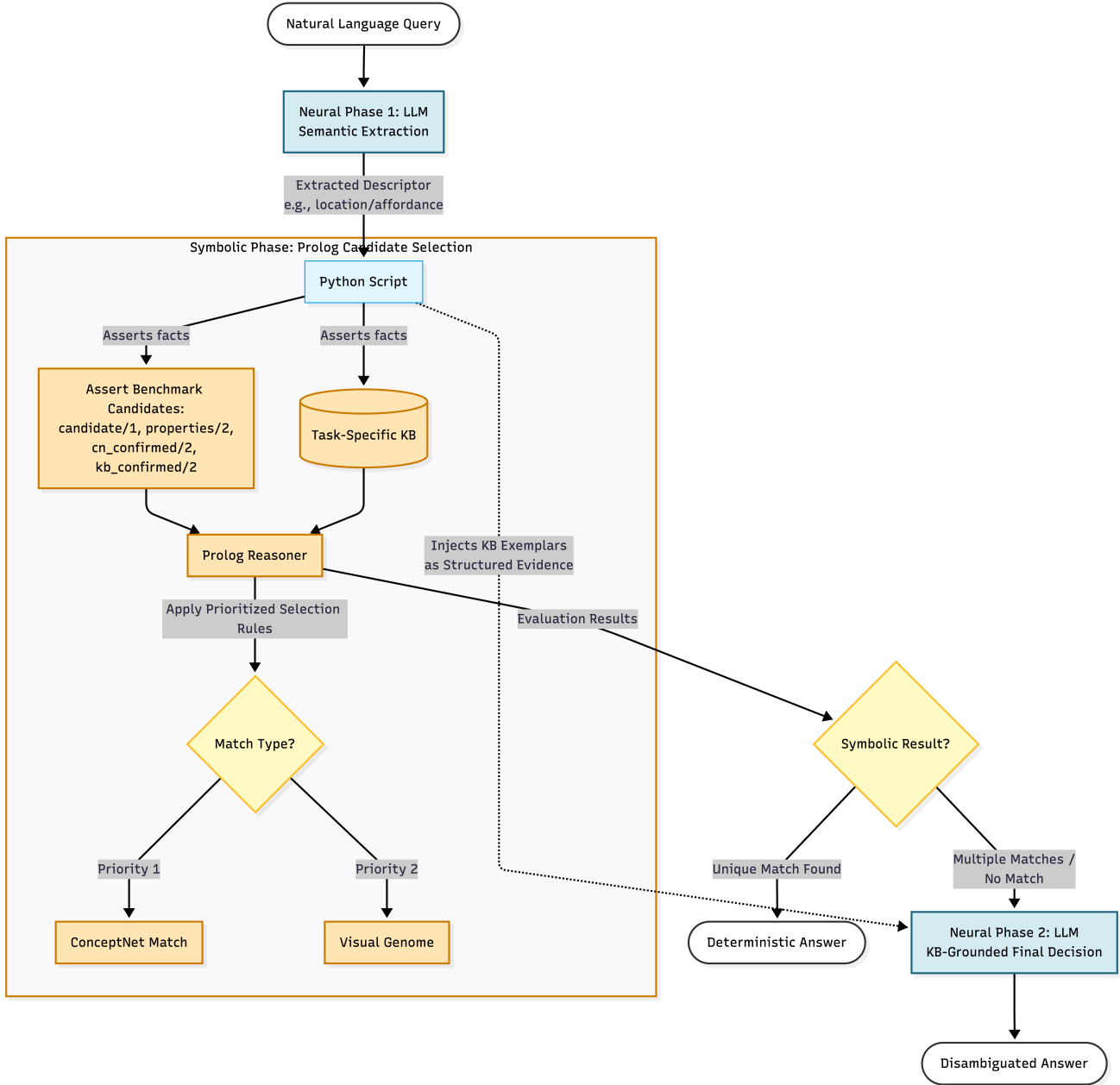


Figure 5.1: Architecture of Family A (KB-Grounded Neuro-Symbolic Loop), illustrating the interaction between the LLM, the knowledge base, and the Prolog reasoner for the *Tidy Up* and *Tool Usage* tasks. Blue boxes mark LLM or Python steps; orange boxes and the cylinder mark Prolog-side operations and the task-specific knowledge base; grey boxes are intermediate data; yellow diamonds are decisions; and rounded white boxes are inputs or final answers. The orange frame denotes the symbolic phase, and dotted arrows show knowledge-base evidence supplied to the final LLM.

5.2. TASK-SPECIFIC IMPLEMENTATIONS

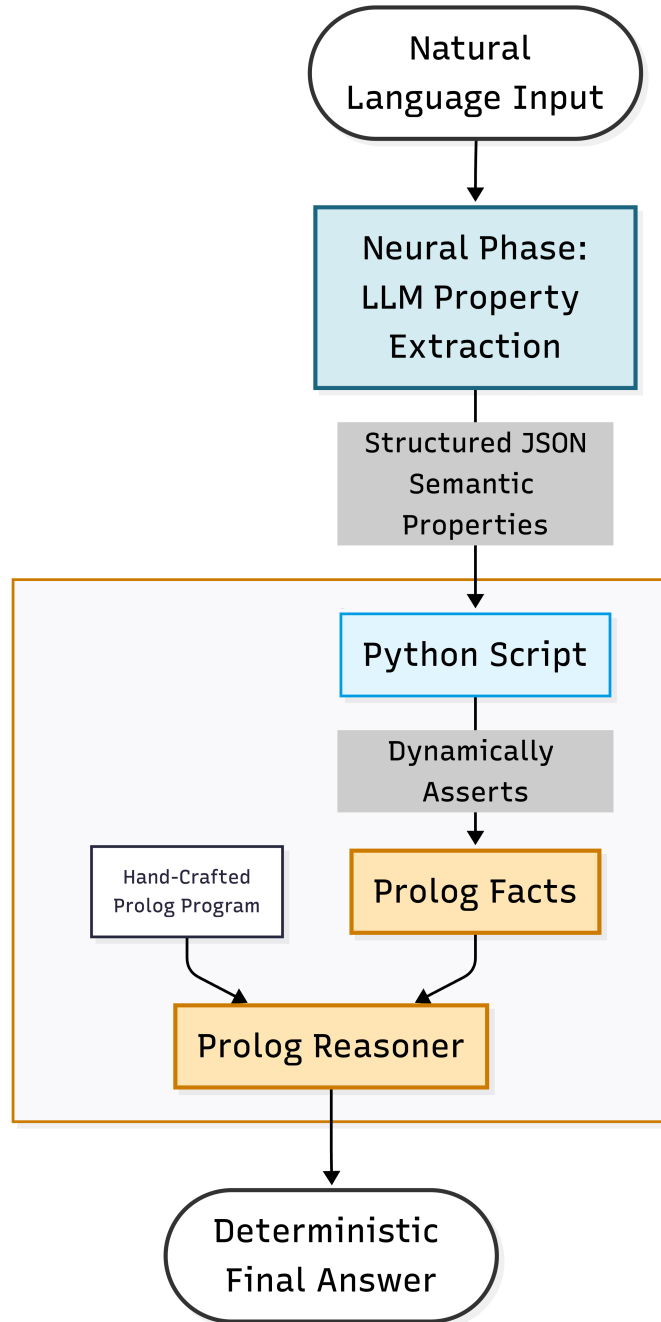


Figure 5.2: Architecture of Family B (Direct LLM-to-Prolog Reasoning), illustrating the simpler two-phase pipeline used for the *Meta-Reasoning*, *Table Setting*, and *Procedural Knowledge* tasks. As in Figure 5.1, blue rectangles denote LLM and Python processing steps, orange rectangles denote Prolog facts and reasoning, grey rectangles denote structured intermediate data, and rounded white boxes denote the input and final answer. The orange enclosure identifies the symbolic phase.

neural extraction step is presented first, followed by the symbolic reasoning phase, with task-specific design choices explained where they arise.

5.2.1 Tidy Up — Location KB and Prolog Location Selector

The *Tidy Up* task requires the system to predict the correct storage location for a household object. It is evaluated in two distinct modalities: a closed-world multi-choice format and an open-ended generative format; the interaction between the neural and symbolic components is tailored accordingly.

Multi-Choice (Closed-World) Format

In the closed-world format, the system must select the correct storage location from a small set of candidate options provided by the benchmark. The pipeline proceeds through three sequential phases:

- **Neural Phase 1 (Semantic Extraction).** The LLM processes the natural-language query and extracts a canonical location type from a 24-entry predefined vocabulary (e.g., kitchen, bathroom), using a synonym normalisation table that maps variant location names to canonical forms — including lowercasing, whitespace standardisation, and resolving lexical variations (e.g., *washroom* → bathroom, *living room* → living_room) (the location extraction prompt is detailed in Appendix A.4.1).
- **Symbolic Phase (Prolog Candidate Selection).** For each candidate location, the Python program asserts four types of Prolog facts (the Prolog candidate selector rules are detailed in Appendix A.5.1):
 - `candidate(Location)` — the candidate’s name;
 - `has_location(Location, Canon)` — its canonical type after synonym normalisation, where Canon refers to the normalised location category (one of 24 predefined labels such as kitchen, bathroom, or garage) obtained by mapping synonymous location names to a single canonical form;
 - `kb_confirmed(Location, Canon)` — asserted only if the knowledge base confirms that the queried object is associated with this location type;
 - `required_canonical(Canon)` — the canonical type inferred by the LLM.

The Prolog reasoner then applies a four-tier priority scheme: (1) a unique KB-confirmed match is selected with confidence `kb_unique`; (2) a unique canonical match (regardless of KB confirmation) is selected with confidence `unique`; (3) if

5.2. TASK-SPECIFIC IMPLEMENTATIONS

multiple candidates match, `SELECTED=ambiguous` is output along with the full matching set; and (4) if no candidate matches, `SELECTED=none` is output. Both `kb_unique` and `unique` result in a deterministic selection without requiring Neural Phase 2; the distinction between them is preserved for downstream confidence logging and evaluation, as a KB-confirmed match carries stronger evidential support than a purely canonical match.

- **Neural Phase 2 (Grounded Disambiguation).** This phase is invoked only in the `ambiguous` and `none` cases (i.e., tiers 3 and 4 above). The LLM is re-invoked with KB exemplar objects associated with the target location injected into the prompt (e.g., “Objects typically found in the kitchen include: pot, pan, spatula, toaster”), providing structured grounding context for the final selection.

Open-Ended (Generative) Format

In the open-ended generative format, no candidate options are provided, and the system must produce and rank the most likely storage locations from scratch. Since there is no closed candidate set, the Prolog reasoner cannot perform direct candidate selection and instead, the knowledge base is leveraged through a hybrid context-injection and re-ranking mechanism:

- **Contextual Hint Injection.** The Python program queries the ConceptNet knowledge base for the target object. If locations are found, up to five normalised typical locations are injected into the LLM prompt as a structured hint (e.g., “Knowledge base hint: ‘mug’ is typically found in: kitchen”).
- **Grounded LLM Generation.** The LLM generates a ranked list of predicted locations informed by both its parametric knowledge and the injected KB context.
- **Symbolic KB Re-ranking.** A normalisation step maps the LLM’s raw predictions to ground-truth-compatible short forms. Any location confirmed by the ConceptNet knowledge base is then promoted to the front of the ranked list, preserving the original KB order, while the remaining LLM-generated locations are appended. This hybrid ranking ensures that verified commonsense locations are prioritised at the top of the list without restricting the generative flexibility of the model.

5.2.2 Tool Usage — Affordance KB and Prolog Tool Selector

The *Tool Usage* task presents only one evaluation format: multi-choice questions in which the system must select the most appropriate tool for a given task description from a set of candidate tools.

- **Neural Phase 1 (Affordance Extraction).** The LLM receives the task description and extracts the single most relevant physical affordance from a canonical vocabulary of 86 labels (e.g., cut, drill, mix) (the affordance extraction prompt is detailed in Appendix A.4.2).
- **Symbolic Phase (Prolog Candidate Selection).** The Prolog interface mirrors the *Tidy Up* reasoner in structure (detailed in Appendix A.5.2). For each candidate tool, the Python driver program (which orchestrates the interaction between the LLM and the Prolog engine) asserts:
 - `candidate(Tool)` — the tool’s name;
 - `has_affordance(Tool, Affordance)` — each affordance associated with the tool in the knowledge base, expanded through 37 manually curated synonym groups (e.g., cut/chop/dice/slit);
 - `cn_confirmed(Tool, Affordance)` — asserted when the affordance link is confirmed by ConceptNet provenance data;
 - `required_affordance(Affordance)` — the affordance extracted by the LLM.

The selection logic is structurally identical to the *Tidy Up* reasoner: it prioritises ConceptNet-confirmed unique matches, falls back to any unique match, and signals ambiguous or none when deterministic selection is not possible. When the symbolic phase cannot resolve a unique answer, the LLM is re-invoked with KB exemplar tools associated with the required affordance injected as grounding context, a mechanism analogous to the disambiguation step described for *Tidy Up* in Section 5.2.1.

5.2.3 Meta-Reasoning — Robot Capability Axioms in Prolog

The *Meta-Reasoning* task assesses whether a given robot hardware configuration can physically execute a given household task. It is evaluated in two formats: a binary classification (can the robot execute the task?) and a multi-choice selection (which of the candidate robot configurations is most appropriate?). Because the relevant features, which are task requirements and robot capabilities, are both fully specified in the input, this task is handled by the simpler Family B pipeline, with no recourse to an external knowledge base.

Neural Phase: Task Requirement Extraction. Given a natural-language task description (e.g., “slice a tomato on a cutting board”), the LLM produces a JSON dictionary with six boolean physical requirements (the extraction prompt is detailed in Appendix A.4.3):

5.2. TASK-SPECIFIC IMPLEMENTATIONS

1. `needs_grasping` — whether the task requires lifting an object completely into the air, as opposed to pushing or sliding.
2. `needs_precision` — whether the task involves highly constrained multi-axis rotations (e.g., inserting a key, threading a needle).
3. `needs_mobility` — whether the robot strictly requires a mobile base, under the assumption that it is always positioned within arm’s reach of all relevant objects.
4. `needs_bimanual` — whether the task absolutely requires two arms working simultaneously with no passive surface support.
5. `needs_contact` — whether the robot must physically touch anything.
6. `needs_rigid_grip` — whether the object is heavy or resistive, requiring a firm rigid grip.

The system prompt contains detailed disambiguation guidelines and six worked examples with chain-of-thought reasoning to calibrate the model’s judgement. Particular care is taken for easily confused properties: for instance, the prompt instructs the model to output false for `needs_bimanual` when stirring a pot, since stirring is inherently a single-arm action that does not require simultaneous manipulation with both hands.

Symbolic Phase: Prolog Capability Matching. The robot’s hardware configuration is encoded as five Prolog facts (`robot_mobile/1`, `robot_arms/1`, `robot_dof/1`, `robot_gripper_type/1` and `robot_rigid/1`), while the task requirements are asserted in a symmetric manner. The Prolog reasoner (detailed in Appendix A.5.3) defines two predicates for binary and multi-choice evaluation respectively:

- `can_execute` (relaxed)—used for binary classification. It checks that the robot satisfies each requirement: a mobile base if mobility is needed, at least one arm with a gripper if grasping is needed, at least 7 degrees of freedom (DoF) if precision is needed, two arms if bimanual manipulation is needed, a rigid grip if rigidity is needed, and a minimum of 3 DoF for any contact task.
- `can_execute_strict` (strict)—used for multi-choice selection, applying tighter thresholds to discriminate between configurations that all satisfy the basic requirements.

An additional `explain_deficiencies` predicate generates a diagnostic certificate listing specific capability failures (e.g., `insufficient_dof(7,4)`, `missing_gripper`),

which supports interpretable error analysis. For multi-choice questions, the candidates that pass the capability check are further ranked by a heuristic score rewarding more arms ($\times 1000$), higher DoF ($\times 100$), the presence of a gripper (+50), and mobility (+30), ensuring that the most capable valid configuration is selected.

5.2.4 Table Setting — Meal Property Extraction and Prolog Cutlery Rules

The *Table Setting* task requires the system to predict the appropriate cutlery set and plate type for a given meal. Ground truth is established through crowdsourced human annotations from 51 annotators: cutlery items endorsed by at least 10 out of 51 annotators are included in the reference set, and the plate type is determined by plurality vote.

Neural Phase: Meal Property Extraction. Given only the meal name, the LLM produces a JSON dictionary with nine boolean culinary properties (the extraction prompt is detailed in Appendix A.4.4): `is_soup_or_broth` (predominantly liquid), `is_handheld_food` (eaten by hand), `needs_cutting` (contains large solid pieces requiring cutting at the table), `is_asian_cuisine` (traditional East/Southeast Asian dish where chopsticks are appropriate), `is_long_noodles` (features long strands of pasta or noodles), `has_sauce_or_small_pieces` (loose scoopable elements such as rice or beans), `is_dessert` (a sweet baked good or dessert), `is_skewer_food` (served on a skewer), and `is_tongs_food` (requires tongs to eat). The system prompt includes seven worked examples addressing common edge cases, such as distinguishing baked casseroles from scoopable bowl meals and classifying Vietnamese sandwiches as handheld rather than chopstick cuisine.

Symbolic Phase: Prolog Cutlery and Plate Rules. The Prolog reasoner (detailed in Appendix A.5.4) implements two complementary rule sets. The cutlery rules map meal properties to a subset of seven possible utensils (Hands, Tongs, Knife, Fork, Skewer, Chopsticks, Spoon): soups trigger Spoon and Bowl; handheld foods trigger Hands and Dinner Plate; meals requiring cutting trigger both Knife and Fork; Asian cuisine triggers Chopsticks, with Fork added for long-noodle dishes; and the default for unlabelled meals is Fork on a Dinner Plate. The plate rules follow a priority cascade: Bowl for soups and scoopable dishes, Dessert Plate for desserts, Coupe Plate for assembled dishes, and Dinner Plate as the default. The Prolog program outputs structured tokens (`CUTLERY=[...]` and `PLATE=...`) that the Python program parses to produce the final prediction.

5.2. TASK-SPECIFIC IMPLEMENTATIONS

5.2.5 Procedural Knowledge — Temporal Property Extraction and Prolog Step Ordering

The *Procedural Knowledge* task tests temporal reasoning about recipe execution. Given a recipe title and two candidate steps, the system must determine which step occurs first. Also in this case we have two different types of questions: the binary evaluation format poses four balanced questions per recipe pair (“Did A occur before B?”, “Did B occur before A?”, “Did B occur after A?”, “Did A occur after B?”), while the multi-choice format asks the system to select the correct predecessor or successor from a list of candidates.

Neural Phase: Step Property Extraction. The LLM receives the recipe title and both steps simultaneously and produces a JSON dictionary with six properties per step (the extraction prompt is detailed in Appendix A.4.5):

1. `cooking_phase` — a categorical label from an 11-phase cooking ontology designed specifically for this thesis to capture the canonical temporal progression of recipe execution: preheating, prep, mixing, marinating, heating, cooking, assembly, resting, cooling, finishing, serving. The ordering of these phases was informed by standard culinary workflows and validated empirically against the benchmark’s recipe corpus.
2. `requires_heat` — whether the step involves applying heat.
3. `is_final_step` — whether the step is one of the very last actions in the recipe (serving, plating, garnishing).
4. `transforms_state` — whether the step fundamentally changes the physical state of the food.
5. `depends_on_other` — whether this step logically requires the other step to have been completed first.
6. `belongs_to_recipe` — whether the step plausibly belongs to the specified recipe, used to filter distractor steps in multi-choice questions.

Symbolic Phase: Prolog Temporal Ordering. The Prolog reasoner (detailed in Appendix A.5.5) implements six prioritised rules for determining whether Step A precedes Step B, applied in decreasing order of evidential strength:

1. **Phase-based ordering** : if the two steps belong to different cooking phases, the step in the earlier phase in the 11-level hierarchy comes first.

2. **Dependency-based ordering:** if Step B explicitly depends on Step A's output, then A comes first.
3. **Reverse dependency:** if Step A depends on Step B, then `comes_before` fails.
4. **Final-step heuristic:** a non-final step always precedes a final step.
5. **Heat progression:** within the same phase, a step not requiring heat precedes one that does.
6. **State transformation:** within the same phase, a step that does not transform the physical state of the food precedes one that does.

A validity gate (`valid_steps`) ensures that both steps belong to the specified recipe before any ordering rule fires, preventing spurious orderings when a distractor step from an unrelated recipe is presented in multi-choice questions.

Chapter 6

Experimental Evaluation

The preceding chapters introduced two neuro-symbolic pipelines that integrate large language models with symbolic reasoning engines, though in fundamentally different ways. Pipeline 1 (Chapter 4) couples an LLM with ILASP-style ASP rule induction, learning symbolic rules from data; Pipeline 2 (Chapter 5) couples it with hand crafted SWI-Prolog programs, encoding domain knowledge by hand. Despite their architectural differences, both pipelines are designed to test a shared hypothesis: whether augmenting neural language models with structured symbolic reasoning can reliably improve commonsense reasoning performance in household robotics.

Whether this hypothesis holds, and under what conditions, cannot be settled by architectural arguments alone. The interaction between neural flexibility and symbolic rigidity is difficult to predict a priori since a knowledge base might help a weak model yet be redundant for a strong one, or a rule system might excel on constrained tasks while hurting performance on open ended ones. Empirical evaluation is therefore essential. Specifically, this chapter addresses RQ1–RQ3, introduced in Chapter 1:

- RQ1** Does a neuro-symbolic pipeline outperform a matched pure-LLM baseline, and how does this effect vary across tasks and answer formats?
- RQ2** To what extent does the symbolic layer compensate for differences in neural model capacity?
- RQ3** What do neural-only and symbolic-only ablations reveal about the contribution and limitations of each component?

The evaluation is primarily conducted on Pipeline 2, which covers all five tasks of the RoboCSKBench [44]. Where Pipeline 1 results are available (Tidy Up and Tool Usage), they are discussed comparatively to illuminate the trade-offs between learned and hand-crafted symbolic reasoning. This overlap is methodologically useful rather than

redundant: Pipeline 2 establishes breadth, while Pipeline 1 tests whether part of the rule-acquisition process can be automated. The chapter is organised as follows: Section 6.1 describes the models, ablation design, and evaluation metrics; Section 6.2 presents the per-task results; Section 6.3 synthesises cross-cutting patterns and Section 6.4 discusses implications, limitations, and design principles.

6.1 Experimental Setup

6.1.1 Models and Configurations

All experiments use models from the Mistral family, accessed through the Mistral API with temperature set to 0.0 to ensure deterministic, reproducible outputs. Three model sizes are evaluated:

- **Mistral Large** (123B parameters, dense architecture; accessed via `mistral-large-latest`): the highest-capacity model, serving as a neural ceiling.
- **Mistral Medium** (parameter count undisclosed by Mistral AI, widely estimated at 64–70B; accessed via `mistral-medium-latest`): an intermediate-capacity model.
- **Mistral Small** (24B parameters, dense architecture; accessed via `mistral-small-latest`): the smallest model, included specifically to test whether the symbolic layer can compensate for limited model capacity.

All models receive the same task-specific system prompt within each task and configuration; the prompt differs across tasks because each requires a different structured output. The complete Pipeline 2 prompts are reproduced in Appendix A.4. The rationale for evaluating three sizes is straightforward: by holding the symbolic component constant and varying only the neural component, we can distinguish between two modes of improvement. If the symbolic layer provides a roughly constant gain regardless of model size, the improvement is *additive* meaning that the symbolic module contributes information that no amount of scaling provides. If, instead, the gain is larger for weaker models and diminishes for stronger ones, the improvement is *compensatory* indicating that the symbolic layer substitutes for the parametric knowledge that a larger model already encodes internally.

6.1.2 Ablation Study Design

To attribute performance to the individual components of the pipeline, we employ a three-way ablation. Each configuration isolates a different contribution, and together they an-

6.1. EXPERIMENTAL SETUP

answer the question of whether improvement stems from neural extraction, symbolic reasoning or their interaction.

Full Pipeline

This configuration consists of the complete neuro-symbolic architecture described in Sections 5.1.1 and 5.1.2. For Family A tasks (Tidy Up, Tool Usage), this includes: (1) LLM semantic extraction, (2) Prolog candidate selection with KB grounding, and (3) LLM disambiguation when Prolog cannot uniquely resolve. For Family B tasks (Meta-Reasoning, Table Setting, Procedural Knowledge), this includes: (1) LLM property extraction and (2) deterministic Prolog rule application.

Pure LLM

The Prolog reasoner and all knowledge-base lookups are bypassed entirely and the LLM directly produces the final answer from the raw input. This ablation establishes the baseline performance achievable through the LLM’s parametric knowledge alone and it serves as a critical reference point because, if the full pipeline fails to exceed this baseline, then it means that the symbolic layer adds no measurable value. Instead, where it does exceed the baseline, the margin quantifies the symbolic layer’s contribution.

Pure Logic

In this case we replace the LLM with a keyword based heuristic extractor. Simple keyword matching maps input tokens to the same property schema that the LLM would produce and the Prolog reasoner operates on these keyword-derived facts. This ablation reveals the Prolog rules’ dependence on the quality of the input they receive: a high Pure Logic score would indicate that the hand crafted rules are well calibrated to the benchmark and that the task’s input vocabulary maps cleanly onto the property schema, making neural semantic understanding unnecessary. On the other hand, a low score would indicate that the rules are only as good as the facts they receive, that semantic extraction is the bottleneck, not symbolic inference.

6.1.3 Benchmark and Metrics

All experiments are performed on the RoboCSKBench [44]. The benchmark includes three zero-shot baselines without symbolic augmentation: GPT-4o [33], Llama-3.3-70B [13], and Gemma-2-27b-it [41]. The five tasks, together with their evaluation formats and metrics, are:

1. **Tidy Up** (383 multi-choice questions; 383 open-ended ranking questions): Multi-choice accuracy (MC) for the closed format; Mean Reciprocal Rank (MRR), MAP@ k , and MAR@ k for $k \in \{1, 3, 5\}$ in the open/generative format.
2. **Tool Usage** (291 multi-choice questions): Multi-choice accuracy.
3. **Meta-Reasoning** (9,576 binary classification questions; 4,788 multi-choice questions): Accuracy, precision, recall, specificity, and F1 for the binary format; accuracy for the multi-choice format.
4. **Table Setting** (100 meals): Plate type accuracy (exact match) and cutlery Jaccard similarity (set overlap between predicted and crowdsourced ground-truth cutlery).
5. **Procedural Knowledge** (3,104 binary questions; 1,600 multi-choice questions): Accuracy, precision, recall, specificity, and F1 for the binary format; accuracy for the multi-choice format.

6.2 Results on RoboCSKBench

This section presents and analyses the in-distribution results, organised by task. For each task, the discussion is structured around three questions: (1) does the symbolic layer help? (2) for which model sizes is the gain largest? (3) what does the ablation profile reveal about the task’s structure?

6.2.1 Tidy Up

Multi-Choice Format

Three patterns stand out in Table 6.1.

Consistent symbolic advantage. Across all three model sizes, the full pipeline exceeds the pure LLM baseline: +9.9 pp for Mistral Large, +8.6 pp for Mistral Medium, and +16.7 pp for Mistral Small. This consistency is underpinned by near-complete KB coverage since 382 of 383 objects (99.7%) have the correct answer present in the KB exemplar set, meaning that the symbolic layer has access to the information it needs in almost every case. That said, coverage alone does not guarantee accuracy. The KB entries were collected from ConceptNet and may encode location associations that differ subtly from the ground truth used to construct the benchmark (e.g., listing “cupboard” where the benchmark expects “cabinet”) and these distributional mismatches account for the residual errors even under full coverage.

6.2. RESULTS ON ROBOCSKBENCH

Table 6.1: Tidy Up multi-choice results (383 questions). The full pipeline outperforms the pure-LLM baseline for all three model sizes, with the largest gain observed for Mistral Small (+16.7 pp).

Model	Approach	Accuracy	Correct / Total
Mistral Large	Full pipeline	0.700	268 / 383
Mistral Large	Pure LLM	0.601	230 / 383
Mistral Medium	Full pipeline	0.538	206 / 383
Mistral Medium	Pure LLM	0.452	173 / 383
Mistral Small	Full pipeline	0.546	209 / 383
Mistral Small	Pure LLM	0.379	145 / 383
None	Pure Logic	0.326	125 / 383
Pipeline 1	Learned ASP rules	0.350	134 / 383
Random choice	Reference	0.200	–
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.509	195 / 383
Llama 3.3	Pure LLM	0.496	190 / 383
Gemma 2	Pure LLM	0.522	200 / 383

Capacity compensation via factual retrieval. Mistral Small receives the largest absolute improvement (+16.7 pp), a pattern consistent with a compensatory interpretation: the knowledge base supplies explicit object-location facts that Mistral Large already partially encodes in its parameters but that Mistral Small lacks. In this context, the symbolic layer functions essentially as a retrieval-augmented external memory that narrows the performance gap across model sizes. We explore the limits of this capacity compensation in Section 6.3.2.

Near-random Pure Logic performance. The Pure Logic ablation achieves only $\sim 32\%$, barely above the random baseline for a multi-choice task. This shows that keyword based extraction is insufficient for Tidy Up because correctly mapping an object like “*wok*” to the canonical location type `kitchen` requires genuine semantic understanding, not surface level keyword matching. Neither the neural nor the symbolic component is sufficient in isolation; the pipeline derives its advantage precisely from their interaction.

Performance relative to published baselines. The best published zero-shot baseline is Gemma 2 at 0.522 accuracy. With the full pipeline, Mistral Large reaches 0.700, an improvement of +17.8 pp over that reference. Even Mistral Small with the full pipeline (0.546) exceeds the three published baselines. This comparison is informative but not fully controlled, since the published baselines use different prompting and output-

normalisation conditions. The stronger evidence for symbolic grounding therefore comes from the within-pipeline ablation, where the full pipeline improves over the matched pure-LLM baseline for every Mistral model.

Open / Generative Format

Table 6.2: Tidy Up open/generative results (383 objects). MRR is the primary metric. The pure-LLM baseline marginally outperforms the full pipeline across all model sizes.

Model	Approach	MRR	MAP@1	MAP@3	MAP@5	MAR@1	MAR@3	MAR@5
Mistral Large	Full pipeline	0.849	0.760	0.831	0.819	0.168	0.259	0.314
Mistral Large	Pure LLM	0.882	0.812	0.872	0.857	0.188	0.290	0.347
Mistral Medium	Full pipeline	0.865	0.791	0.842	0.810	0.165	0.233	0.273
Mistral Medium	Pure LLM	0.897	0.836	0.878	0.842	0.177	0.256	0.296
Mistral Small	Full pipeline	0.726	0.616	0.685	0.703	0.138	0.205	0.247
Mistral Small	Pure LLM	0.739	0.632	0.707	0.721	0.141	0.221	0.264
Pipeline 1	Learned ASP rules	0.761	0.655	0.554	0.543	0.146	0.255	0.302
<i>Published RoboCSKBench baselines (zero-shot)</i>								
GPT-4o	Pure LLM	0.412	0.345	0.387	0.365	0.147	0.221	0.249
Llama 3.3	Pure LLM	0.424	0.364	0.392	0.372	0.158	0.225	0.256
Gemma 2	Pure LLM	0.276	0.218	0.255	0.248	0.096	0.144	0.157

The open/generative format inverts the multi-choice finding since the Table 6.2 shows that the pure LLM baseline achieves marginally higher MRR than the full pipeline across all model sizes. This reversal has a methodological explanation. In the open format, the system must generate and rank a free form list of plausible locations and there is no closed candidate set. The unconstrained LLM produces broader ranked lists that are rewarded by the MRR metric, whereas the symbolic re-ranking mechanism, which promotes KB confirmed locations to the top, occasionally displaces a correct LLM generated location in favour of a KB confirmed alternative that the benchmark ground truth does not rank first.

Taken together, the multi-choice and open-format results expose a structural property of symbolic grounding: its primary value lies in precise discrimination among closed candidate sets, not in generative breadth. When the task requires selecting one correct answer from a constrained option set, the KB provides decisive grounding but, when the task is open ended, the LLM’s parametric knowledge yields more diverse outputs.

Baseline comparisons. A notable feature of Table 6.2 is the large gap between all of our configurations and the published zero-shot baselines. RoboCSKBench reports MRR values of 0.424 (Llama 3.3), 0.412 (GPT-4o), and 0.276 (Gemma 2). Even our weakest setup (Mistral Small with the full pipeline) achieves an MRR of 0.726, and our strongest

6.2. RESULTS ON ROBOCSKBENCH

configuration (Mistral Medium, Pure LLM) reaches 0.897. But, again, we cannot directly compare the two approaches due to the differences in prompt engineering techniques used.

Pipeline 1 results (ILASP rule induction). Pipeline 1 learns ASP rules offline via ILASP using `mistral-small-latest` and is evaluated on unseen test objects from the benchmark.

In the **open/generative format**, the full pipeline achieves an MRR of 0.7611 with an Any-Hit Accuracy of 97.61%. Precision is strongest at $k=1$ (0.6552) and declines modestly at larger cut-offs (0.5535 at $k=3$, 0.5428 at $k=5$), while recall grows from 0.1461 at $k=1$ to 0.3017 at $k=5$. These figures suggest that the offline-induced rules transfer to unseen items in this setting, providing useful semantic structure even in unconstrained generation. They do not establish broader generality by themselves, since the induced rules are evaluated within the same benchmark task family.

In the **multi-choice format** (383 questions), the learned rules of Pipeline 1 achieve an accuracy of 34.99% (134/383 correct) using `mistral-medium-latest`.

Comparing the two pipelines reveals a clear trade-off between the offline-learned rules of Pipeline 1 and the handcrafted cascades of Pipeline 2. In the open-ended generation format under the same neural backbone (`mistral-small-latest`), Pipeline 1 outperforms Pipeline 2, achieving a higher open-format MRR (0.7611 vs. 0.726) and a higher recall at $k=5$ (0.3017 vs. 0.247), though Pipeline 2 retains a slight edge in average precision at $k=5$ (0.703 vs. 0.681). However, this performance gap in the open format is primarily a methodological artifact of the MRR metric. Under Pipeline 1’s offline-induced rules (which are relatively general), Clingo frequently fires multiple rules, retrieving a larger and more diverse set of candidate locations for each object. Since MRR and recall metrics reward the presence of any correct location anywhere within the retrieved list, this broader generative retrieval behavior naturally inflates Pipeline 1’s open-format scores. Conversely, this lack of specificity is a primary factor in Pipeline 1’s weak performance on the closed-set multi-choice task. When evaluated on the multi-choice format (383 questions) using the same backbone (`mistral-medium-latest`), Pipeline 2’s handcrafted cascades significantly outperform Pipeline 1 (53.8% vs. 34.99%). A diagnostic analysis of Pipeline 1 supports this interpretation: Clingo provided location hints in 298 cases (77.8%), but these logic-guided choices yielded an accuracy of only 40.6% (121 correct, 177 wrong). In the remaining 85 cases (22.2%), no rules fired, forcing the model to fall back on LLM-only selection, which achieved a mere 15.3% accuracy (13 correct, 72 wrong). This suggests that while the induced rules fire frequently, they remain to solve the fine grained candidate choices. Pipeline 2’s handcrafted Prolog cascades provide more task-specific heuristic specificity, but this advantage is partly purchased through manual

rule engineering.

6.2.2 Tool Usage

Table 6.3: Tool Usage multi-choice results (291 questions). The KB-grounded pipeline produces the largest absolute improvement observed in any task, with Mistral Small gaining +41.9 pp.

Model	Approach	Accuracy	Correct / Total
Mistral Small	Full pipeline	0.818	238 / 291
Mistral Small	Pure LLM	0.399	116 / 291
Mistral Large	Full pipeline	0.814	237 / 291
Mistral Large	Pure LLM	0.581	169 / 291
Mistral Medium	Full pipeline	0.735	214 / 291
Mistral Medium	Pure LLM	0.588	171 / 291
None	Pure Logic	0.285	83 / 291
Pipeline 1	Learned ASP rules	0.399	116 / 291
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.529	154 / 291
Llama 3.3	Pure LLM	0.522	152 / 291
Gemma 2	Pure LLM	0.526	153 / 291

Tool Usage provides the strongest evidence for the value of symbolic grounding in this evaluation. The results in Table 6.3 reveal a phenomenon not observed in any other task since the smallest model, when equipped with the full pipeline, matches the largest model.

Mistral Small achieves 81.8% accuracy with the full pipeline but only 39.9% without it, a gain of +41.9 pp, representing the largest single improvement recorded across all tasks and configurations. Mistral Large with the full pipeline reaches 81.4%, a negligible 0.4 pp difference. Yet the pure LLM gap between Mistral Small (39.9%) and Mistral Large (58.1%) is 18.2 pp, indicating that the knowledge base eliminates this gap entirely.

The practical implication is that in domains where task-critical factual associations can be reliably enumerated in a curated knowledge base, augmenting a smaller, cheaper model with symbolic retrieval can match the performance of a larger unaugmented model. However, this shift in the cost-performance trade-off is strictly limited to tasks that are heavily bottlenecked by factual retrieval rather than complex reasoning.

Why is Tool Usage so responsive to KB grounding? The answer lies in the nature of the knowledge required. Selecting the correct tool demands knowing that a *hammer* pos-

6.2. RESULTS ON ROBOCSKBENCH

sesses the *hammering* affordance, that a *wrench* affords *fastening*, or that a *chisel* affords *carving*. These are factual associations between specific objects and specific physical capabilities, precisely the kind of knowledge that a curated knowledge base encodes directly and that an LLM encodes only indirectly through distributional patterns in its training corpus. In the pure-LLM ablation, the model must perform both implicit knowledge retrieval and the reasoning step in a single forward pass; the neuro-symbolic pipeline decomposes these responsibilities, delegating retrieval to the KB and selection logic to the Prolog reasoner.

Published baselines. All three model sizes with the full pipeline exceed the published zero-shot baselines by wide margins. The full pipeline reaches 0.818 accuracy (Mistral Small) and 0.814 (Mistral Large), compared with the best published baseline of 0.529 (GPT-4o). As in Tidy Up, this cross-system comparison is affected by differences in prompting and answer handling. The matched ablation remains the decisive comparison: on the same 291 questions, adding KB-grounded symbolic selection raises Mistral Small from 0.399 to 0.818 and Mistral Large from 0.581 to 0.814.

Pipeline 1 results (ILASP rule induction). Pipeline 1 achieves a tool selection accuracy of only 39.86% (116/291) on the same 291 multi-choice questions, far below Pipeline 2’s 81.8%. Examining the symbolic resolution statistics clarifies why: Clingo finds a unique tool match in only 15.1% of cases, returns multiple candidates (requiring LLM tie-breaking) in 57.7%, and finds no match at all in 27.1%, at which point the system falls back to direct LLM selection. The underlying affordance prediction accuracy is 38.8%, rising to 58.1% when synonym expansion is enabled suggesting that SOMA feature annotation coverage is the primary bottleneck.

The gap between the two pipelines on Tool Usage (39.86% vs. 81.8%) is substantially larger than on Tidy Up, and the reason is structural. Pipeline 1 is highly rigid: it attempts to extract SOMA features for every candidate tool on the fly and match them using offline-induced rules. If the LLM makes any annotation error or omission (such as failing to tag a tool with a key physical quality or affordance), the symbolic path fails entirely.

Pipeline 2, by contrast, adopts a reverse-lookup architecture that is more robust. Instead of performing direct rule-matching on candidate tool annotations, it first extracts the required affordance from the task (e.g., *absorb*) and queries the curated knowledge base to retrieve a list of known exemplar objects (e.g., *sponge*, *cloth*). These exemplars are then supplied to the LLM as a contextual hint, allowing the neural component to leverage its broad semantic representation to map the candidate choices to the functionally similar exemplars (e.g., selecting *paper towel* because it behaves like *sponge*). Additionally,

Pipeline 2 uses semantic synonym mappings (e.g., expanding absorb to dry) to ensure exemplars are successfully retrieved even when the exact affordance term differs.

6.2.3 Meta-Reasoning

Binary Classification

Table 6.4: Meta-Reasoning binary classification results (9,576 questions). Pure Logic achieves the highest F1 (0.996), indicating near-perfect calibration of the hand-crafted Prolog axioms to the in-distribution benchmark.

Model	Approach	Accuracy	Precision	Recall	Specificity	F1
Mistral Large	Full pipeline	0.973	1.000	0.945	1.000	0.972
Mistral Large	Pure LLM	0.767	1.000	0.533	1.000	0.696
Mistral Medium	Full pipeline	0.971	1.000	0.942	1.000	0.970
Mistral Medium	Pure LLM	0.708	0.997	0.418	0.999	0.589
Mistral Small	Full pipeline	0.971	1.000	0.942	1.000	0.970
Mistral Small	Pure LLM	0.795	0.990	0.596	0.994	0.744
None	Pure Logic	0.996	1.000	0.993	1.000	0.996
<i>Published RoboCSKBench baselines (zero-shot)</i>						
GPT-4o	Pure LLM	0.772	0.994	0.546	0.997	0.705
Llama 3.3	Pure LLM	0.943	0.988	0.897	0.989	0.940
Gemma 2	Pure LLM	0.907	0.961	0.848	0.965	0.901

The pattern in Table 6.4 requires careful interpretation. That Pure Logic achieves $F1 = 0.996$ does *not* mean that the LLM is unnecessary in general, it means that the hand crafted Prolog capability axioms are almost perfectly calibrated to the specific task-robot pairings in the RoboCSKBench. The benchmark’s task descriptions contain explicit action verbs (“slice”, “bring”, “arrange”) that map directly to the boolean property schema (needs_grasping, needs_cutting, etc.), allowing even the keyword extractor to produce adequate property vectors.

The full pipeline ($F1$ ranging from 0.970 for Mistral Small and Medium to 0.972 for Mistral Large) sits between Pure Logic and the pure LLM baselines (ranging from 0.589 to 0.744). Interestingly, Mistral Small Pure LLM achieves an $F1$ of 0.744, outperforming the larger models in the pure LLM configuration, though it still falls far short of the full pipeline. The LLM’s primary failure mode in the pure LLM ablation is a high false-negative rate since it tends to underestimate the capabilities of simple robot configurations, conservatively declaring tasks infeasible that the Prolog axioms correctly identify as executable.

6.2. RESULTS ON ROBOCSKBENCH

Why does the full pipeline score lower than Pure Logic? The LLM extraction step occasionally misclassifies a property (e.g., marking `needs_bimanual` as true for a task that can be performed with one arm), and this extraction error propagates through the Prolog reasoner, flipping a correct prediction to incorrect. The keyword extractor, paradoxically, makes *fewer* errors on these well structured task descriptions because it maps keywords deterministically. One interpretation is that, on these unusually explicit descriptions, the LLM extraction step introduces distinctions that the benchmark’s keyword-aligned property schema does not reward.

Context from published baselines. Among the zero-shot baselines reported in the benchmark, Llama 3.3 achieves the strongest binary F1 (0.940), followed by Gemma 2 (0.901) and GPT-4o (0.705). Both our Pure Logic baseline (0.996) and the full pipeline (0.972 for Mistral Large) exceed even Llama 3.3 by substantial margins. This supports the conclusion that, for this benchmark’s binary capability checks, explicit Prolog axioms are more reliable than direct parametric judgment. The result should not be read as evidence that the same axioms would dominate on every robot-capability dataset without further OOD validation.

Multi-Choice Format

Table 6.5: Meta-Reasoning multi-choice results (4,788 questions). Pure Logic achieves the highest accuracy (0.943).

Model	Approach	Accuracy	Correct / Total
Mistral Large	Full pipeline	0.925	4427 / 4788
Mistral Large	Pure LLM	0.913	4370 / 4788
Mistral Medium	Full pipeline	0.929	4447 / 4788
Mistral Medium	Pure LLM	0.917	4390 / 4788
Mistral Small	Full pipeline	0.933	4465 / 4788
Mistral Small	Pure LLM	0.892	4273 / 4788
None	Pure Logic	0.943	4516 / 4788
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.790	3783 / 4788
Llama 3.3	Pure LLM	0.737	3529 / 4788
Gemma 2	Pure LLM	0.658	3151 / 4788

The multi-choice format (Table 6.5) shows the same pattern at smaller margins: Pure Logic leads at 0.943, the full pipeline clusters at 0.925–0.933, and Pure LLM ranges from 0.892 to 0.917. The gap compression reflects the greater ease of multi-choice questions where even the pure-LLM baseline exceeds 89% by leveraging the explicit specifications

in the candidate descriptions. For context, the published zero-shot baselines achieve 0.790 (GPT-4o), 0.737 (Llama 3.3), and 0.658 (Gemma 2). Our weakest full-pipeline configuration (Mistral Large, 0.925) still exceeds GPT-4o by +13.5 pp. This is consistent with a benefit from rule-based constraint enforcement, although the comparison to published baselines again combines different models, prompts, and output procedures.

6.2.4 Table Setting

Table 6.6: Table Setting results (100 meals). The full pipeline consistently outperforms both ablations on both metrics.

Model	Approach	Plate Acc.	Cutlery Jaccard
Mistral Small	Full pipeline	0.880	0.762
Mistral Large	Full pipeline	0.850	0.778
Mistral Medium	Full pipeline	0.840	0.777
Mistral Large	Pure LLM	0.830	0.748
Mistral Small	Pure LLM	0.820	0.708
Mistral Medium	Pure LLM	0.770	0.585
None	Pure Logic	0.710	0.674
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.730	0.702
Llama 3.3	Pure LLM	0.790	0.739
Gemma 2	Pure LLM	0.790	0.642

Table Setting occupies a middle ground in the ablation spectrum: the full pipeline helps consistently but the gains are moderate compared to Tool Usage or Meta-Reasoning. The largest improvement appears in cutlery prediction for Mistral Medium, where the Prolog rules boost Jaccard similarity from 0.585 (pure LLM) to 0.777, a gain of +19.2 pp. Plate accuracy gains are more modest: +6.0 pp for Mistral Small, +7.0 pp for Mistral Medium, and +2.0 pp for Mistral Large.

Baseline capabilities and extraction variance. The results show that even in the pure LLM configuration, both Mistral Large (0.830) and Mistral Small (0.820) achieve high plate accuracy. This indicates that identifying the correct type of plate or bowl for a given dish is a relatively straightforward commonsense task for modern LLMs, even without symbolic guidance. The moderate improvements of the full pipeline (ranging from +2.0 pp to +7.0 pp) show that while the Prolog cascades provide minor corrections, the primary bottleneck is not complex reasoning but rather minor extraction variance. The slight edge of Mistral Small with the full pipeline (0.880 vs. 0.850 for Mistral Large) is

6.2. RESULTS ON ROBOCSKBENCH

likely a minor statistical fluctuation in how the models extract the boolean food attributes (e.g., classifying a dish as soup or noodle-based) rather than a systematic structural advantage.

Why is the cutlery gain larger than the plate gain? Cutlery prediction requires set-valued output: a meal might require {Fork, Knife, Spoon} simultaneously. The Prolog rules codify complex interactions between properties (e.g., Asian cuisine plus long noodles implies Chopsticks; non-Asian handheld food implies Hands but also Fork and Knife for the full table setting). Producing such set valued outputs consistently in a single LLM forward pass is difficult, and omissions are common. The Prolog rules enforce systematic completeness, ensuring that culturally expected items are not overlooked.

Baselines. The best published zero-shot baseline is Llama-3.3-70B at 0.790 plate accuracy and 0.739 cutlery Jaccard. Our full pipeline with Mistral Small (0.880 plate, 0.762 cutlery) and Mistral Large (0.850 plate, 0.778 cutlery) exceeds these figures on both metrics, but this should be treated as contextual evidence rather than a fully controlled comparison. The internal ablations show the clearer pattern: Prolog priority cascades provide small plate corrections and larger gains for set-valued cutlery prediction.

6.2.5 Procedural Knowledge

Binary Step Ordering

Table 6.7: Procedural Knowledge binary results (3,104 questions). The full pipeline and pure-LLM baseline achieve comparable performance. Pure Logic fails near-randomly (≈ 0.48).

Model	Approach	Accuracy	F1
Mistral Small	Pure LLM	0.983	0.983
Mistral Large	Full pipeline	0.968	0.968
Mistral Small	Full pipeline	0.964	0.964
Mistral Large	Pure LLM	0.960	0.960
Mistral Medium	Full pipeline	0.950	0.950
Mistral Medium	Pure LLM	0.927	0.927
Any model	Pure Logic	0.481	0.481
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.926	0.929
Llama 3.3	Pure LLM	0.915	0.919
Gemma 2	Pure LLM	0.740	0.793

Procedural Knowledge is the task where the neuro-symbolic pipeline adds the least value. In the binary format (Table 6.7), the full pipeline achieves 0.968 accuracy (Mistral Large) while the pure LLM baseline reaches 0.960, a marginal +0.8 pp gain. More notable is that Mistral Small achieves the highest overall accuracy (0.983) in the *pure LLM* configuration, whereas its full-pipeline configuration drops to 0.964, meaning that for this model size, the symbolic constraints actually degrade performance compared to the unconstrained LLM.

This is the one task where a strong LLM can outperform the neuro-symbolic pipeline on the in-distribution benchmark. While one explanation is that LLMs have absorbed strong sequential priors from millions of recipe texts in their training corpora, another plausible explanation is that the handcrafted 11-phase symbolic cooking ontology is simply too coarse to capture the nuanced temporal dependencies of recipes. When the formal ontology fails to adequately model the task domain, enforcing its rules degrades performance relative to the LLM’s unconstrained internal representations.

Pure Logic shows the neural component’s importance. The near-random performance of Pure Logic (0.481) shows that the temporal Prolog reasoner cannot function without accurate phase labels from the LLM. Keywords alone cannot distinguish “mixing” from “cooking” or “prep” from “assembly”, these require genuine semantic understanding of the step descriptions. The Prolog phase hierarchy (preheating < prep < mixing < ... < serving) provides useful structural scaffolding, but without accurate phase classification the scaffolding produces near-chance predictions.

Multi-Choice Format

The multi-choice format (Table 6.8) compounds the binary format’s challenge with recipe specific step plausibility: the model must not only order two steps but also identify which candidate step belongs to the recipe at all. Here the pure LLM baseline marginally outperforms the full pipeline across all model sizes. The LLM’s parametric knowledge about specific recipes (e.g., that searing precedes braising in a pot roast) provides an advantage that the 11-level Prolog phase hierarchy cannot replicate, because the hierarchy encodes *generic* temporal structure without recipe specific semantics.

Relation to published baselines. On binary classification, our configurations exceed all published baselines: Mistral Small Pure LLM (0.983) surpasses even GPT-4o (0.926) by +5.7 pp, although prompt and parsing differences again limit the strength of this comparison. The multi-choice format tells a different story. Our best result (Mistral Medium Pure LLM, 0.863) surpasses Llama 3.3 (0.848) and Gemma 2 (0.809) but falls short of

6.3. CROSS-TASK ANALYSIS

Table 6.8: Procedural Knowledge multi-choice results (1,600 questions). The pure-LLM baseline marginally outperforms the full pipeline.

Model	Approach	Accuracy	Correct / Total
Mistral Medium	Pure LLM	0.863	1380 / 1600
Mistral Medium	Full pipeline	0.856	1369 / 1600
Mistral Large	Pure LLM	0.849	1359 / 1600
Mistral Large	Full pipeline	0.841	1345 / 1600
Mistral Small	Pure LLM	0.801	1282 / 1600
Mistral Small	Full pipeline	0.798	1277 / 1600
Any model	Pure Logic	0.665	1064 / 1600
<i>Published RoboCSKBench baselines (zero-shot)</i>			
GPT-4o	Pure LLM	0.914	1462 / 1600
Llama 3.3	Pure LLM	0.848	1357 / 1600
Gemma 2	Pure LLM	0.809	1294 / 1600

GPT-4o’s 0.914.

6.3 Cross-Task Analysis

The per-task results presented above exhibit markedly different performance profiles. This section steps back from individual tasks and identifies the cross-cutting patterns that emerge from the full evaluation.

6.3.1 Conditions for Symbolic Advantage

Table 6.9: Summary of the neuro-symbolic advantage across all tasks (using Mistral Large, in-distribution). Δ measures the full pipeline’s gain over the pure-LLM baseline.

Task	Format	Full Pipeline	Pure LLM	Δ
Tool Usage	MC	0.814	0.581	+23.3 pp
Meta-Reasoning	Binary	0.973	0.767	+20.6 pp
Tidy Up	MC	0.700	0.601	+9.9 pp
Table Setting	Plate	0.850	0.830	+2.0 pp
Table Setting	Cutlery	0.778	0.748	+3.0 pp
Proc. Knowledge	Binary	0.968	0.960	+0.8 pp
Proc. Knowledge	MC	0.841	0.849	−0.8 pp
Tidy Up	Open	0.849	0.882	−3.3 pp

Table 6.9 arranges the results by the magnitude of the neuro-symbolic advantage, and a clear gradient emerges. To ensure a controlled and consistent comparison across all

task types, the results in this table are standardized on Mistral Large, the largest evaluated model, representing the upper-bound performance of both configurations. At the top of the table sit the tasks where the symbolic layer provides the largest gains: Tool Usage (+23.3 pp), Meta-Reasoning binary (+20.6 pp), and Tidy Up MC (+9.9 pp). What these tasks share is a dependence on *factual knowledge retrieval*: the correct answer hinges on specific object-affordance, object-location or robot-capability associations that the KB encodes directly but that the LLM encodes only indirectly.

The middle of the table contains Table Setting, where the Prolog rules provide modest but consistent improvements (+2.0 pp and +3.0 pp on Mistral Large). Here the symbolic contribution is less about factual retrieval and more about *structural constraint enforcement*: the priority cascade and set valued cutlery rules impose a systematic completeness that the LLM struggles to achieve in a single forward pass.

At the bottom, two task-format combinations show a slight advantage for the pure-LLM baseline: Tidy Up Open (−3.3 pp) and Procedural Knowledge MC (−0.8 pp), representing two distinct failure modes of symbolic grounding. In the open-ended format, the symbolic re-ranking mechanism trades off generative breadth for precision, occasionally displacing correct but KB-unconfirmed predictions. In procedural multi-choice, the symbolic rules encode generic temporal structure (an 11-level phase hierarchy) that might be too coarse to capture recipe specific knowledge, exactly the kind of fine grained semantics where the LLM’s parametric memory excels.

A third cross-cutting pattern is the correlation between the symbolic layer’s benefit and model size. Across all Family A tasks, the absolute gain of the full pipeline over the pure-LLM baseline is largest for Mistral Small and smallest for Mistral Large, consistent with the compensatory interpretation introduced earlier.

6.3.2 The Equalisation Effect

Table 6.10: The equalisation effect: the full pipeline compresses the performance gap between model sizes. Δ_{LLM} is the pure-LLM gap between Mistral Large and Small; Δ_{Full} is the full-pipeline gap.

Task	Δ_{LLM}	Δ_{Full}	Gap Reduction
Tool Usage	18.2 pp	0.4 pp	97.8%
Tidy Up (MC)	22.2 pp	15.4 pp	30.6%
Table Setting (Plate)	1.0 pp	−3.0 pp	100%+

Table 6.10 quantifies what we term the *equalisation effect*, i.e. the degree to which the symbolic layer compresses the performance gap between large and small models. The most striking case is Tool Usage, where the 18.2 pp pure-LLM gap between Mistral

6.4. DISCUSSION

Large and Small shrinks to 0.4 pp with the full pipeline, a 97.8% reduction. For Table Setting plate accuracy, the gap inverts entirely since Mistral Small with the full pipeline *outperforms* Mistral Large with the full pipeline.

We restrict the equalization analysis in Table 6.10 to these three task-format configurations because they are the only settings that satisfy two preconditions simultaneously: (1) the neuro-symbolic pipeline achieves a positive performance gain ($\Delta > 0$) over the pure LLM baseline (which excludes Tidy Up Open and Procedural Knowledge MC, where the symbolic layer does not outperform the baseline), and (2) there is a substantial initial positive capacity gap between Mistral Large and Small under the pure LLM baseline (which excludes Table Setting Cutlery, Meta-Reasoning MC, Meta-Reasoning Binary, and Procedural Knowledge Binary, where the initial pure LLM gaps are either very small or negative due to Mistral Small matching or outperforming Mistral Large in the unconstrained setup).

This finding suggests that for tasks with structured constraints where symbolic grounding is effective, there are direct implications for system deployment. In such settings, when latency, cost, or other constraints favor smaller models, the neuro-symbolic architecture can provide a mechanism to recover a substantial portion of the performance that would otherwise be lost by downsizing the neural component. However, this equalisation is not universal and remains limited to configurations where the symbolic layer’s rules and constraints align closely with the task requirements.

6.4 Discussion

6.4.1 Limitations of the Experimental Design

Several limitations constrain the generality of the conclusions drawn from this evaluation.

Model family coverage. All primary experiments use the Mistral model family. A more comprehensive evaluation spanning architecturally diverse models (e.g., GPT-4, Claude, or Gemini) would strengthen the claim that the observed advantages are architecture-level rather than family-specific.

Pipeline 1 vs. Pipeline 2. The comparison between Pipeline 1 (ILASP-based rule induction) and Pipeline 2 (hand-crafted SWI-Prolog rules) is limited to Tidy Up and Tool Usage. On these two tasks, Pipeline 1 suggests that automatically induced rules can match or exceed hand crafted ones for location assignment, but it underperforms significantly on tool selection due to annotation sensitivity. This highlights a fundamental trade-off: Pipeline 1 automates rule acquisition from data, reducing the engineering effort required,

but its accuracy is bounded by the quality of the input features. Improving the SOMA annotation part and also scaling the rule induction to support more expressive hypothesis schemas is a natural direction for future work.

KB construction bias. The knowledge base used by Family A tasks was constructed from ConceptNet, which encodes predominantly Western commonsense associations. Cultural biases in the KB (e.g., location conventions for kitchenware) may inflate performance on benchmarks that share these assumptions and deflate it on cross-cultural datasets.

6.4.2 Design Principles for Neuro-Symbolic Systems

The experimental results point to several design principles for systems that combine LLMs with symbolic reasoners.

Symbolic reasoning for constraint enforcement, not knowledge generation. The Prolog rules excel when they enforce domain-invariant structural constraints such as phase ordering, capability thresholds and culinary conventions but they struggle when they must capture instance specific knowledge. Knowledge bases, not rules, should serve as the vehicle for factual associations.

Property extraction over answer generation. The separation of concerns where LLM extracts properties and Prolog derives conclusions consistently outperforms end-to-end LLM reasoning across all tasks except those where the LLM’s parametric knowledge is already near-optimal (Procedural Knowledge). Designing the LLM’s role as a *perception layer* rather than a *decision layer* constrains its output space, reduces hallucination risk and allows the symbolic module to enforce guarantees that neural generation alone cannot provide.

Invest in the knowledge base when factual retrieval is the bottleneck. The equalisation effect suggests that a well-curated KB paired with a small model can match or exceed a large model operating without symbolic support on specific factual-association tasks such as Tool Usage. For deployment-constrained scenarios where latency, cost, or on-device requirements limit model size, this suggests that KB construction and curation can be a competitive alternative to model scaling. The Procedural Knowledge results show the boundary of this principle: when performance depends on nuanced instance-specific semantics rather than recoverable factual associations, a larger or stronger LLM may still be the more effective component.

6.4. DISCUSSION

Beyond accuracy: explainability, auditability, and safety guarantees. While the quantitative results show task-dependent performance advantages, a neuro-symbolic pipeline also introduces significant non-functional benefits over pure LLM baselines. First, it offers *explainability and traceability*: in a pure-LLM architecture, the final decision is derived from a black-box forward pass, offering no trace of the underlying reasoning. In our pipeline, the decision is computed by a symbolic engine (e.g., Prolog or Clingo), allowing the system to output exact logical proof traces detailing which rules were triggered and which KB facts were retrieved. Second, this separation makes the system more *auditable and correctable*: if the robot makes a mistake, developers can immediately determine whether the error lies in the perception layer (e.g., the LLM misextracting a property) or the reasoning layer (e.g., a missing KB fact or incorrect Prolog rule), enabling targeted fixes. Finally, it can provide hard *safety and constraint guarantees* for the constraints that are explicitly formalised: physical robotic controllers can deploy symbolic rule solvers to programmatically block actions that violate encoded capability thresholds or environmental constraints. These guarantees are only as complete as the rule base and the extracted facts supplied to it.

Chapter 7

Generalization Study

The previous chapter evaluated the neuro-symbolic pipelines on the benchmark for which they were designed. That evaluation identified conditions under which symbolic reasoning helps, but it left open a central question: do these gains persist on data outside the design distribution?

A pipeline that excels on its home benchmark may do so for benchmark-specific reasons: the Prolog rules may match the annotation conventions, the knowledge base may reflect the same commonsense distributions as the ground truth, or the prompts may be well calibrated to the benchmark’s vocabulary. The goal of this chapter is to investigate whether the advantages observed in Chapter 6 transfer beyond RoboCSKBench or whether they are primarily artefacts of benchmark-pipeline alignment.

This chapter addresses the fourth research question introduced in Chapter 1:

RQ4: *To what extent does the runtime neuro-symbolic pipeline generalise to out-of-distribution data without task-specific adaptation?*

Answering this question requires distinguishing two forms of generalisation. The first is *same-structure transfer*: the pipeline encounters new objects, tasks or recipes, but the underlying task structure is preserved so, for example, the system still selects a location from a candidate set or still temporally orders two recipe steps. The second is *cross-structure transfer*: the pipeline encounters a task whose answer structure differs from what it was designed for, for example, selecting the correct physical *procedure* rather than the correct physical *entity*. The distinction matters because same-structure transfer tests whether the symbolic rules encode genuinely general constraints, while cross-structure transfer tests whether the pipeline’s architectural assumptions are portable.

To conduct this investigation, five out-of-distribution (OOD) datasets were assembled from independent academic sources, covering four of the five tasks evaluated in Chapter 6. Table Setting is excluded because no suitable external dataset with comparable plate and

7.1. OUT-OF-DISTRIBUTION DATASETS

cutlery annotations could be identified. The pipeline code, Prolog rules, knowledge base, and LLM prompts were applied *without modification*, following a strict zero-shot transfer protocol. Dataset-side preprocessing, such as vocabulary normalisation for simulator-specific names, is documented explicitly and treated as a potential source of bias.

7.1 Out-of-Distribution Datasets

A significant practical challenge of this evaluation was the scarcity of off-the-shelf benchmarks that match the task structure of RoboCSKBench. Most OOD datasets required manual construction from raw data sources, a process documented for each dataset below. No suitable OOD dataset was found for Table Setting, which is therefore excluded from this chapter. Each OOD dataset was created to stress a specific aspect of the pipeline’s generalisation capacity. Table 7.1 summarises the five datasets; the remainder of this section describes the construction methodology and the nature of the distributional shift for each.

Table 7.1: Summary of OOD datasets used in the generalisation study. All datasets are evaluated with the unmodified Pipeline 2 code. “Shift type” indicates the primary axis of distribution shift relative to RoboCSKBench.

Dataset	Task Family	Size	Format	Source	Shift Type
VirtualHome	Tidy Up	67	MC	[35]	Vocabulary
TidyBot	Tidy Up	598	MC	[46]	Personalised prefs.
PIQA	Tool Usage	710	MC	[6]	Task structure
BEHAVIOR-1K	Meta-Reas.	2,032	Binary	[22]	Activity vocabulary
RecipeNLG	Proc. Know.	2,000	Binary	[5]	Writing style

7.1.1 VirtualHome

VirtualHome [35] is a simulation platform for household activities that provides a structured inventory of household objects, their placement destinations, and semantic flags identifying which entities can be grasped or used as containers and surfaces.

The dataset construction script filters objects to those flagged as GRABBABLE in the Unity property dictionary and destinations to those flagged as CONTAINERS or SURFACES, excluding non-storage furniture (beds, sofas, rugs). Object and destination names are normalised through a manual synonym mapping layer (e.g., `wine_glass` → `wineglass`, `cutting_board` → `cuttingboard`) to bridge the gap between the placing script’s natural-language names and Unity’s internal class identifiers. The resulting dataset contains 67 five-way multiple-choice questions. For each object, the most frequent valid destination

is treated as the correct answer; four distractors are sampled without replacement from the remaining destination vocabulary, excluding the object’s other recorded destinations.

The distributional shift in this case is primarily *lexical*. VirtualHome uses compound or camelCase identifiers (`kitchencabinet`, `AlarmClock`) that do not match the natural language tokens used in ConceptNet or the RoboCSKBench vocabulary. This creates a vocabulary bottleneck for the knowledge base lookup since, for example, the query `has_location(kitchencabinet, ...)` finds no matching node in ConceptNet unless a normalisation step splits and lowercases the identifier first. To isolate this bottleneck, a vocabulary-alignment layer normalises VirtualHome names to KB-compatible tokens before the Prolog lookup. This preprocessing improves comparability with RoboCSKBench, but it also means that the VirtualHome results should be read as transfer under explicit lexical normalisation rather than fully raw zero-shot transfer.

7.1.2 TidyBot

While VirtualHome provides a well-structured evaluation set that preserves the same answer structure as the RoboCSKBench Tidy Up task, its small size (67 questions) limits statistical power. TidyBot was therefore added as a complementary, larger-scale OOD source for the same task family. TidyBot [46] is a real-world robotic tidying benchmark in which a mobile manipulator must return household objects to their intended receptacles. The benchmark’s distinguishing feature is *personalised placement preferences*, meaning that different users in different scenarios may prefer different receptacles for the same object. The evaluation dataset was constructed from the benchmark’s held-out object–receptacle assignments, which specify placements that the robot must infer at test time.

For each unique object encountered across all scenarios, a five-way multiple-choice question is generated: the correct receptacle from the first scenario in which the object appears, plus four distractors sampled from the global receptacle vocabulary (14 unique receptacles across all scenarios) for a total of 598 multiple-choice questions.

The distributional shift relative to RoboCSKBench is twofold: (1) TidyBot’s objects include fine-grained household items (specific brands, informal names) that may not appear in ConceptNet; and (2), more importantly, the *correct* placement is determined by individual user preferences, not by universal commonsense. A “magazine” might belong on the coffee table for one user and in the recycling bin for another, challenging the pipeline’s implicit assumption that object-location mappings are universal factual associations.

7.1. OUT-OF-DISTRIBUTION DATASETS

7.1.3 PIQA

PIQA [6] (Physical Intuition Question Answering) is a benchmark for physical common-sense reasoning. Each question presents a goal (e.g., “To keep your fridge smelling fresh”) and two candidate solutions, of which exactly one is physically plausible. The OOD dataset was constructed by filtering the PIQA validation split (1,838 questions) to retain only household-related goals, using Mistral Small as a binary classifier with the prompt: “Is this goal household-related? Reply yes or no.” This yielded 710 domain-relevant questions. Because this filtering step is itself model-assisted, the filtered subset should be interpreted as an operational household subset rather than a manually certified taxonomy. Unlike the BEHAVIOR-1K annotations described below, this filter does not create or alter target labels: PIQA’s original correct answers are retained, and results are also reported on the complete validation split. A separate annotation model was therefore not required to establish the ground truth, although model-assisted selection may still bias the composition of the household subset.

This time, the distributional shift is *structural*, not merely lexical. The RoboCSK-Bench Tool Usage task requires selecting the correct *tool* (a named entity) for a given action. PIQA requires selecting the correct *procedure* (a free-text description) for a given goal. The pipeline’s affordance-based KB lookup retrieves object-level affordance triples, for example:

$$\text{hammer} \xrightarrow{\text{UsedFor}} \text{hammering}$$

but PIQA’s correct answers are procedural descriptions, not named objects. This represents a cross-structure transfer since the pipeline’s entity-selection architecture is applied to a procedure-selection task.

7.1.4 BEHAVIOR-1K

BEHAVIOR-1K [22] defines 1,000 everyday activities spanning 50 activity categories in simulated household environments. For the Meta-Reasoning OOD evaluation, 250 diverse tasks were sampled from 10 categories, and each task was annotated with minimum hardware requirements using a local Llama 3.1 (8B) model. A different model family (Llama) was chosen for annotation to avoid architecture overlap with the Mistral models used at test time, which could inflate results through shared representational biases. The LLM classifies each task along five axes: mobility, number of arms, degrees of freedom, gripper type, and gripper rigidity. For each task, a “minimum viable robot” configuration that satisfies all requirements is generated as a positive example and a mutated configuration that violates at least one critical requirement is generated as a negative example. This yields 2,032 balanced binary classification questions (1,016 positive, 1,016 negative). To

validate annotation quality, a random subset of 50 task–configuration pairs ($\approx 20\%$ of the 250 tasks) was manually inspected; all checked annotations were found to be correct, though this limited sample size means residual annotation errors cannot be ruled out.

The distributional shift is primarily in *activity vocabulary*. BEHAVIOR-1K introduces task descriptions such as “caulk the bathtub seam” or “assemble the IKEA bookshelf” that are absent from RoboCSKBench’s curated task inventory. The task descriptions use more varied and informal language, and the activity categories (maintenance, repair, outdoor) extend well beyond RoboCSKBench’s focus on indoor manipulation and table-level tasks.

7.1.5 RecipeNLG

RecipeNLG [5] is a large-scale corpus of cooking recipes scraped from web sources. The OOD evaluation dataset was constructed by extracting consecutive step pairs, retaining at most two candidate pairs per recipe, and then sampling 500 pairs. The final sample contains one pair from each of 500 distinct recipes and generates 2,000 balanced binary questions via the same four-way question template used for RoboCSKBench (“Did step A occur before step B?”, “Did step B occur before step A?”, and the corresponding “after” variants). Here the distributional shift is the source corpus rather than a presumed difference in editorial quality. Both RecipeNLG and the Recipe1M+ source used by RoboCSKBench contain web-derived recipes and may include informal or noisy instructions. RecipeNLG nevertheless supplies previously unseen recipes, wording, and step combinations, allowing the evaluation to test transfer across independently collected cooking data. The Prolog temporal reasoner must handle these inputs using the same 11-phase hierarchy (preheating < prep < mixing < ... < serving) without any domain-specific adaptation.

7.2 Results

All OOD evaluations use the unmodified Pipeline 2 (LLM $\rightarrow$ Prolog) with the same ablation structure as Chapter 6: Full Pipeline, Pure LLM, and Pure Logic and, also in this case, the tests for multiple model sizes are reported. The evaluation protocol is strict zero-shot transfer: no prompts, rules, or KB entries were modified for the OOD datasets.

7.2.1 Tidy Up: VirtualHome and TidyBot

The VirtualHome results in Table 7.2 present a clear ceiling effect: all models achieve above 88% accuracy, with the pure-LLM baseline marginally outperforming the full pipeline by 1.5 pp for Mistral Large and Medium. This pattern is consistent with the hypothesis that VirtualHome’s object–location associations (e.g., beer in the fridge, alarm clock on

7.2. RESULTS

Table 7.2: Tidy Up OOD accuracy on VirtualHome (67 questions) and TidyBot (598 questions). Bold marks the highest accuracy within each dataset.

Dataset	Model	Full Pipeline	Pure LLM	Δ
VirtualHome	Mistral Large	91.04%	92.54%	−1.5 pp
	Mistral Medium	92.54%	94.03%	−1.5 pp
	Mistral Small	88.06%	88.06%	0.0 pp
	Pure Logic		49.25%	
TidyBot	Mistral Large	47.99%	39.13%	+8.9 pp
	Mistral Medium	41.30%	30.94%	+10.4 pp
	Mistral Small	34.45%	26.15%	+8.3 pp
	Pure Logic		21.40%	

the nightstand, plate in the kitchen cabinet) are high-frequency commonsense associations that LLMs encode reliably from pre-training, so the KB lookup adds no information that the LLM does not already possess, and the pipeline’s additional processing steps introduce a small probability of error through candidate reranking.

Critically, however, the evaluation set is very small (67 questions), which means that the 1.5 pp delta corresponds to a single additional missed question. No statistical significance can be claimed for a difference of this magnitude at this sample size; the result is consistent with noise.

Pure Logic accuracy (49.25%) is notable in a different respect since, on a 5-way multiple-choice task, random chance is 20%, so 49.25% represents $2.46\times$ the random baseline. This is the highest Pure Logic accuracy observed on any multiple-choice task in either the in-distribution or OOD evaluations, suggesting that VirtualHome’s clean, structured data (derived from JSON inventory files) provides an unusually favourable input for keyword-based matching. Nevertheless, Pure Logic alone remains far below the LLM configurations, confirming that semantic interpretation is still the dominant factor.

TidyBot tells a different story since absolute accuracy is low across all configurations in Table 7.2, with the best result being 47.99%, well below the in-distribution Tidy Up MC accuracy of 70.0%. This reflects the fundamental challenge of personalised placement preferences: the “correct” receptacle for a water bottle might be the desk in one household and the recycling bin in another, and no amount of commonsense knowledge can disambiguate between culturally valid alternatives.

The symbolic layer nonetheless provides consistent gains: +8.9 pp for Mistral Large, +10.4 pp for Mistral Medium and +8.3 pp for Mistral Small. The gain persists because the KB provides a useful default even when the exact answer is ambiguous. Concept-Net’s *AtLocation* triples encode the *prototypical* location for household objects, which correlates with the most common TidyBot preference even if it does not match every indi-

vidual user. The pipeline effectively uses commonsense as a soft bias toward prototypical placements, and this bias appears to be more informative than the LLM’s unaided guess.

Pure Logic accuracy (21.40%) supports the same pattern observed in-distribution: the symbolic rules are ineffective without the LLM’s semantic extraction. The 5.3 pp gap between Pure Logic and random chance (20% for a 5-way MC question) indicates that even keyword matching captures a small residual signal, but not enough to be practically useful.

7.2.2 Tool Usage: PIQA

Table 7.3: PIQA OOD results. The domain-filtered subset (710 Q) retains only household-related goals. The full pipeline underperforms the pure-LLM baseline in both subsets.

Subset	Full Pipeline	Pure LLM	Δ
Filtered (710 Q)	91.69%	94.37%	−2.7 pp
Unfiltered (1,838 Q)	82.37%	88.37%	−6.0 pp
Pure Logic Baseline (Model-Independent)		49.30% (Filtered)	48.48% (Unfiltered)

Table 7.3 presents the clearest negative result in the generalisation study. Unlike the other OOD tasks, only Mistral Large is evaluated here. The primary analytical interest is the comparison between the domain-filtered and unfiltered subsets, which isolates the impact of ConceptNet coverage under a structural task shift. Mistral Large was chosen to establish an upper bound on performance; however, we note that evaluating additional model sizes would strengthen the claim that the observed deficit is architectural rather than parametric.

The full pipeline underperforms the pure-LLM baseline by 2.7 pp on the filtered subset and 6.0 pp on the unfiltered set while pure Logic hovers at 49%, indistinguishable from random chance on a binary task.

The reason is architectural: the Tool Usage pipeline was designed for *entity selection*: it retrieves affordance triples from ConceptNet (e.g., *hammer* → *UsedFor:hammering*), counts how many retrieved exemplars mention each candidate answer and promotes the candidate with the highest KB-mention differential. This mechanism assumes that the correct answer is a *named entity* whose affordances are catalogued in the knowledge base, but PIQA violates this assumption. Its correct answers are free-text procedural descriptions (“place a cup of vinegar on the top shelf”), not named objects. The KB-mention differential penalises creative correct answers that do not mention KB-indexed tool names and rewards incorrect answers that happen to reference familiar objects. A wrong answer mentioning “screwdriver” or “pliers” may accumulate KB mentions that a correct answer

7.2. RESULTS

phrased as a procedure will not, systematically biasing the pipeline toward entity rich but factually incorrect candidates.

The 6.0 pp penalty on the unfiltered set (compared to 2.7 pp on the filtered set) provides a second piece of evidence for this explanation. The unfiltered set includes non-household goals (e.g., automotive repair, crafting) where KB coverage is sparser, amplifying the differential bias.

7.2.3 Meta-Reasoning: BEHAVIOR-1K

Table 7.4: BEHAVIOR-1K OOD results (2,032 binary questions, derived from 250 sampled tasks and balanced positive/negative robot configurations). The full pipeline achieves the highest F1 across all model sizes, with the largest gain for Mistral Large (+29.2 pp F1).

Model	Approach	Acc.	Prec.	Rec.	Spec.	F1
Mistral Large	Full pipeline	0.770	0.691	0.976	0.564	0.809
Mistral Large	Pure LLM	0.667	0.943	0.356	0.978	0.517
Mistral Medium	Full pipeline	0.765	0.701	0.924	0.606	0.797
Mistral Medium	Pure LLM	0.681	0.905	0.405	0.958	0.559
Mistral Small	Full pipeline	0.768	0.695	0.955	0.581	0.804
Mistral Small	Pure LLM	0.747	0.855	0.595	0.899	0.702
Pure Logic baseline (model-independent)		0.625	0.572	0.992	0.257	0.725

Table 7.4 provides the strongest evidence for generalisation in the study. The full pipeline outperforms the pure-LLM baseline across all three model sizes, with the most pronounced improvement for Mistral Large: F1 rises from 0.517 (pure LLM) to 0.809 (full pipeline), a gain of +29.2 pp and the largest single delta observed in the OOD evaluation.

The pattern is particularly instructive when contrasted with the in-distribution results. On RoboCSKBench, Pure Logic alone achieved $F1 = 0.996$, and the full pipeline’s advantage over pure LLM was already large (+27.6 pp). On BEHAVIOR-1K, Pure Logic drops to $F1 = 0.725$, the pure-LLM baseline drops to 0.517, and the full pipeline settles at 0.809, remaining the strongest ablation by a wide margin.

This pattern is consistent with the following explanation. The Prolog capability axioms encode domain-invariant physical constraints like: a task requiring bimanual manipulation needs at least two arms, a task requiring precision manipulation needs at least 6 DoF, a task requiring a firm grip needs a rigid end-effector. These constraints are *answer-structure-invariant*, meaning that they hold regardless of whether the task is “slice a tomato” or “caulk the bathtub seam”. The LLM’s role is to *translate* the novel task

description into the boolean property schema that the Prolog axioms can process. On in-distribution data, even keyword matching suffices for this translation (hence Pure Logic’s 0.996) while, on OOD data, the LLM’s semantic understanding becomes essential because the task vocabulary has shifted.

The pure-LLM baseline’s drop on BEHAVIOR-1K ($F1 = 0.517$ for Mistral Large) exposes the failure mode that the symbolic layer mitigates. The LLM’s primary error is a high false-negative rate: it conservatively declares tasks infeasible that the Prolog axioms correctly identify as executable. Recall drops from 0.945 (in-distribution full pipeline) to 0.356 (OOD pure LLM), while precision remains high (0.943). The LLM appears to hedge when it encounters unfamiliar task descriptions, defaulting to caution. The Prolog axioms instead apply fixed threshold checks and identify executability whenever the extracted physical requirements are met.

The Pure Logic baseline’s deceptively high F1-score (0.725) must be interpreted with caution, as it exposes a significant generalization pathology. In the OOD task vocabulary of BEHAVIOR-1K, the simple keyword-based extractor often fails to match the new task verbs onto the property schema, causing most task requirements (e.g., `needs_grasping`, `needs_precision`) to default to false. Because the symbolic reasoner receives empty requirement profiles, it classifies nearly all task-robot pairings as executable, producing a near-perfect recall (0.992) but a severely degraded specificity (0.257) due to a flood of false positives. This baseline does not represent a functional heuristic but rather a default-true pathology, indicating that semantic dictionary alignment is a critical bottleneck for keyword-based generalisation.

7.2.4 Procedural Knowledge: RecipeNLG

Table 7.5: RecipeNLG OOD results (2,000 binary questions, 500 recipes). The full pipeline outperforms the pure-LLM baseline for all model sizes, reversing the in-distribution pattern where pure LLM led marginally.

Model	Full Pipeline	Pure LLM	$\Delta F1$
Mistral Large	0.898	0.786	+0.112
Mistral Medium	0.812	0.640	+0.172
Mistral Small	0.846	0.798	+0.048
Pure Logic Baseline (Model-Independent)		0.406	

RecipeNLG provides the most instructive results in the generalisation study because it inverts the in-distribution findings. On RoboCSKBench, pure LLM achieved $F1 = 0.983$ (Mistral Small), outperforming the full pipeline (0.968 for Mistral Large). On Table 7.5

7.3. CROSS-DATASET ANALYSIS

shows that the relationship reverses on RecipeNLG: the full pipeline (0.898, Mistral Large) outperforms pure LLM (0.786) by +11.2 pp F1.

This reversal is consistent with a shift in the step combinations presented to the model, but the present experiment cannot attribute it to cleaner or noisier writing. Both RecipeNLG and Recipe1M+, the source used by RoboCSKBench, are web-derived corpora and can contain informal or imperfect instructions. The defensible conclusion is narrower: performance changes on recipes sampled from an independent corpus, where the pure LLM’s learned culinary associations do not transfer as reliably as they do on the in-distribution questions.

The Prolog temporal reasoner, by contrast, does not rely on distributional priors. Its 11-phase hierarchy (preheating < prep < mixing < ... < serving) and its dependency-based ordering rules encode *structural* constraints about cooking processes that hold regardless of writing style. A step classified as “prep” comes before a step classified as “cooking” whether the recipe is for pad thai or duck confit, and whether the instructions are written in formal English or abbreviated shorthand. The LLM still provides essential value since it classifies each step into the correct phase, a task that keyword matching cannot perform reliably. Notably, Pure Logic achieves $F1 = 0.406$ on RecipeNLG, which is *below* the 0.50 random baseline for a balanced binary task. This means the keyword-based phase classifier is not merely uninformative but systematically anti-correlated with the ground truth since it assigns phase labels that produce inverted orderings more often than correct ones. This anti-correlation suggests that RecipeNLG’s informal, abbreviated step descriptions are adversarial for keyword-based parsing, and that the LLM’s semantic phase classification is essential to anchor the Prolog rules in correct step interpretations.

7.3 Cross-Dataset Analysis

Table 7.6 distils the results into a single comparative view that reveals two contrasting patterns.

Pattern A: symbolic advantage preserved or amplified. Three OOD datasets: BEHAVIOR-1K, RecipeNLG and TidyBot preserve or amplify the full pipeline’s advantage over pure LLM. These three share the common property that the OOD task retains the same *answer structure* as the in-distribution task. BEHAVIOR-1K still asks “can this robot execute this task?” (binary classification over physical capabilities). RecipeNLG still asks “which step comes first?” (binary temporal ordering). TidyBot still asks “where does this object go?” (entity selection from a candidate set). In all three cases, the symbolic component, whether Prolog axioms or KB lookup, operates on the same type of structured input it was

Table 7.6: In-distribution vs. OOD performance comparison. Δ_{ID} is the full pipeline’s advantage over pure LLM on RoboCSKBench; Δ_{OOD} is the same on the OOD dataset. All deltas compare Mistral Large in both conditions to ensure a consistent baseline across rows. Note that different tasks use different primary metrics (Accuracy or F1), as indicated by the “Metric” column; cross-row magnitude comparisons should be made with this caveat in mind.

Task and OOD dataset	Metric	Δ_{ID}	Δ_{OOD}	Pattern
Meta-Reas. (BEHAVIOR-1K)	F1	+27.6 pp	+29.2 pp	Amplified
Proc. Know. (RecipeNLG)	F1	+0.8 pp	+11.2 pp	Inverted
Tidy Up (TidyBot)	Acc	+9.9 pp	+8.9 pp	Preserved
Tidy Up (VirtualHome)	Acc	+9.9 pp	−1.5 pp	Eliminated
Tool Usage (PIQA)	Acc	+23.3 pp	−2.7 pp	Inverted

designed for, and the domain-invariant constraints it encodes (physical thresholds, phase orderings, location associations) transfer because they are properties of the *world*, not of the benchmark.

The most striking case is RecipeNLG, where the symbolic advantage *increases* from +0.8 pp in-distribution to +11.2 pp OOD. This inversion occurs because the in-distribution LLM performance was near-ceiling (0.983), leaving no room for the symbolic layer to contribute. On the independently sampled RecipeNLG questions, the Prolog phase hierarchy instead provides a stable ordering constraint.

Pattern B: symbolic advantage eliminated or inverted. Two OOD datasets, VirtualHome and PIQA, eliminate or invert the full pipeline’s advantage. These two differ from the Pattern A datasets in a critical respect since the pipeline’s architectural assumptions are misaligned with the OOD task.

For VirtualHome, the issue is a performance ceiling combined with small-sample sensitivity rather than structural misalignment: the LLM baseline already achieves near-perfect performance (92.54% for Mistral Large) on common household associations, and because the evaluation set contains only 67 questions, even a single additional missed question by the pipeline translates to a −1.5 pp delta. As noted in Section 7.2.1, this difference is not statistically significant. For PIQA, instead, the misalignment is structural: the pipeline’s entity-selection mechanism actively harms performance when applied to a procedure-selection task, because the KB-mention differential biases predictions toward entity-rich but factually incorrect candidates.

The discriminating factor. The results suggest that the key predictor of OOD transfer success is not dataset size, domain proximity, or model capacity, but rather *answer-*

7.4. DISCUSSION

structure alignment: whether the OOD task preserves the same answer type (entity selection, binary classification, temporal ordering) as the in-distribution task. We emphasise that this is a post-hoc observation derived from the five OOD datasets studied here, not a pre-registered hypothesis; further evaluation on additional task families would be needed to confirm its generality. In the datasets studied here, preserved answer structure allows the symbolic rules to operate on the same kind of input for which they were designed. When the answer structure changes (entity selection $\rightarrow$ procedure selection), the symbolic component’s inductive biases can become liabilities.

7.4 Discussion

The OOD evaluation across VirtualHome, TidyBot, PIQA, BEHAVIOR-1K and RecipeNLG shows that neuro-symbolic transfer is highly conditional. The symbolic layer generalises best when the OOD task preserves the same answer structure and when the encoded rules correspond to physical or procedural constraints. It is less reliable when the shift changes the type of answer the system must select or when success depends on personalised preferences that are not represented in the knowledge base.

7.4.1 When and Why Symbolic Rules Generalise

The primary driver of successful OOD transfer is the nature of the knowledge encoded by the symbolic layer. The results suggest a distinction between two types of symbolic knowledge:

- **Physical and logical invariants:** Rules that formalise constraints of the physical world, such as robot hardware capability thresholds in BEHAVIOR-1K or temporal phase orderings in RecipeNLG, transfer comparatively well in zero-shot settings. Because these rules represent properties of the environment rather than statistical regularities of a specific dataset, they function as a robust anchor. When the neural model’s confidence degrades under domain shift, these invariants prevent physically implausible predictions, leading to substantial gains (e.g., +29.2 pp F1 on BEHAVIOR-1K and +11.2 pp on RecipeNLG).
- **Factual and lexical associations:** Knowledge-base triples (e.g., ConceptNet locations) generalize more unevenly because they are bound by the lexical overlap between the dataset and the KB. In tasks like Tidy Up, location priors provide a steady advantage only when the evaluation objects align with the KB’s pre-existing vocabulary.

7.4.2 Systemic Bottlenecks and Limitations

Conversely, when the symbolic layer fails to improve performance or actively degrades it, the root cause falls into one of three systemic categories:

1. **Answer-structure mismatch:** When the core pipeline’s scoring mechanisms assume a specific answer structure (e.g., entity selection), applying the system to a different task structure (e.g., PIQA’s procedure selection) leads to systematic bias. Here, the symbolic layer’s inductive biases become liabilities, as the model selects candidates based on entity frequency rather than procedural correctness.
2. **Lexical bottlenecks:** Symbolic rules are brittle to representation details. In VirtualHome, compound simulator-specific names (e.g., `kitchencabinet`) fail to match KB nodes, forcing the system to fall back on ungrounded neural predictions. This highlights that lexical normalization is a critical dependency for symbolic reasoning.
3. **Ceiling effects and sample sensitivity:** When the LLM baseline already achieves near-ceiling performance on common commonsense associations, as in VirtualHome, the symbolic layer has little headroom to show quantitative improvement. In small evaluation sets, the percentage metric is also highly sensitive: one additional error in VirtualHome changes accuracy by roughly 1.5 pp. Interpretability may still be valuable in such cases, but it should be treated as a separate benefit from accuracy.

7.4.3 Implications for Hybrid Architectures

The results of this chapter suggest three practical implications for the design of future neuro-symbolic systems in physical reasoning domains:

1. **Scope symbolic constraints to physical invariants:** The strongest OOD gains occur when symbolic rules formalise domain-invariant physical or temporal constraints (BEHAVIOR-1K, RecipeNLG), while softer commonsense associations (ConceptNet locations) provide smaller, vocabulary-dependent improvements (TidyBot). This suggests that symbolic engineering effort is often best invested in rules that encode properties of the physical world rather than soft distributional regularities.
2. **Treat vocabulary alignment as a core component:** A symbolic engine is only as good as its grounding layer. Vocabulary normalisation should be treated as an active, first-class component, potentially using LLMs to dynamically translate

7.4. *DISCUSSION*

simulator-specific entities into canonical KB concepts, rather than relying on static dictionaries.

3. **Prioritise OOD validation:** The in-distribution evaluations in Chapter 6 produced ceiling effects that masked the relative strengths and weaknesses of the pipeline's components. OOD evaluation is essential for distinguishing genuine architectural advantages from artefacts of benchmark-pipeline co-design.

Chapter 8

Conclusion

8.1 Summary of the Work

This thesis examines whether a neuro-symbolic approach can match or improve upon LLM-only performance on embodied commonsense tasks while exposing the encoded knowledge and rule applications used in its symbolic derivations. This second aim concerns traceability and transparent reasoning: it makes the decision procedure available for inspection, but does not by itself establish that every premise is correct or that users will find the derivation interpretable. Performance and component contributions are evaluated on RoboCSKBench, whose five tasks require different forms of background knowledge for household robotics. Rather than treating the presence of a symbolic component as an advantage by definition, the evaluation compares complete pipelines with matched pure-LLM and pure-logic configurations and then tests the runtime architecture on external datasets.

8.2 Answers to the Research Questions

Performance relative to LLM-only baselines

Regarding RQ1 (“Does a neuro-symbolic pipeline outperform a matched pure-LLM baseline, and how does this effect vary across tasks and answer formats?”), the neuro-symbolic approach does not produce a uniform improvement over the pure LLM because its contribution depends on what the task requires and on how the answer is represented. The strongest benefits appear when the symbolic component can retrieve a relevant fact or enforce a clear constraint. Tool Usage is the clearest example: with Mistral Large, Pipeline 2 improves accuracy by 23.3 percentage points because the knowledge base supplies explicit affordance information that the model otherwise has to recall from its parameters.

8.2. ANSWERS TO THE RESEARCH QUESTIONS

The mechanism is less useful for broad generation. In open-ended Tidy Up, symbolic re-ranking reduces MRR by 3.3 points when a knowledge-base-supported location displaces another valid answer. RQ1 therefore has a conditional answer: explicit knowledge and rules help only when their representation fits the task and its evaluation format.

Compensation for model capacity

Regarding RQ2 (“To what extent does the symbolic layer compensate for differences in neural model capacity?”), the symbolic layer sometimes reduces the difference between larger and smaller language models, although this effect is concentrated in tasks dominated by factual retrieval. Again, Tool Usage provides the strongest evidence: an 18.2-point gap between Mistral Large and Mistral Small in the pure-LLM condition falls to 0.4 points when both models use the full pipeline. Once the same affordance knowledge is explicitly available, model size has little influence on the final result.

The pattern is weaker elsewhere and disappears near a performance ceiling or when the symbolic representation omits relevant distinctions. A suitable knowledge base can therefore compensate for lower model capacity on some tasks, but symbolic reasoning cannot generally replace the interpretation supplied by a stronger model.

Contribution of the neural and symbolic components

Regarding RQ3 (“What do neural-only and symbolic-only ablations reveal about the contribution and limitations of each component?”), the ablations show that the two components address different failure modes. Without symbolic support, the LLM remains strong on familiar procedural questions but loses the benefit of explicit factual retrieval and capability constraints. Without neural extraction, the logic programs depend on keyword matches that do not transfer reliably to unfamiliar language. On BEHAVIOR-1K, for example, missed keywords produce incomplete task descriptions and cause the pure-logic system to classify too many robot configurations as executable.

The components are complementary, but not independently reliable. Neural interpretation connects varied language to the symbolic vocabulary, while the derivation remains conditional on that interpretation. RQ3 thus reveals a reciprocal dependency: logic can make a decision procedure explicit, but cannot correct a false or missing premise by itself.

Out-of-distribution generalisation

Regarding RQ4 (“To what extent does the runtime neuro-symbolic pipeline generalise to out-of-distribution data without task-specific adaptation?”), generalisation beyond Robo-

CSKBench is similarly dependent on alignment between the pipeline and the new task. The most successful transfer occurs on BEHAVIOR-1K, where the hardware constraints retain their meaning despite unfamiliar activity descriptions and improve F1 by 29.2 points for Mistral Large. PIQA provides the contrasting case. Its answers are free-text procedures rather than named tools, so the entity-selection mechanism inherited from Tool Usage no longer matches the decision being made; the full pipeline consequently trails the pure LLM by 2.7 points on the household subset.

General location knowledge remains useful on TidyBot but cannot fully represent personal preferences, while VirtualHome is limited by a small sample and a near-ceiling baseline. Overall, transfer is most successful when the symbolic component encounters the answer structure it expects. This is a post-hoc explanation rather than a general law, but it is the clearest answer to RQ4 supported by these experiments.

The comparison between the pipelines adds one qualification to these findings. Pipeline 1 automates part of rule construction and produces useful open-ended Tidy Up rankings, but it is less reliable for closed-set candidate selection, especially in Tool Usage. Since the comparison covers only two tasks and Pipeline 1 relies on a restricted representation, it does not establish that manually written rules are generally superior. It shows instead that the construction of symbolic knowledge matters as much as the logic formalism.

8.3 Limitations

The main limitation follows from the same design choice that makes the pipelines inspectable. Before a logic program can reason about an instruction, the situation has to be expressed through a controlled vocabulary. This intermediate representation makes the decision process easier to follow, but it also determines what the system is able to notice. If a graph entry is missing, if a term is mapped to the wrong predicate, or if the LLM extracts an incomplete description of the task, the symbolic component does not have access to the situation that was intended. It reasons over the premises it receives. Pipeline 1 is affected by this problem through annotation noise and through the restrictions imposed on the rules it can learn. Pipeline 2 avoids rule induction, but its hand-crafted programs still reflect design decisions about which conditions matter and how conflicts should be resolved. For this reason, the thesis makes knowledge engineering and semantic grounding more visible, but it does not remove them.

The evidence should also be read within the scope of the experiments conducted here. The main ablations use Mistral models, and the direct comparison between the two pipelines is limited to Tidy Up and Tool Usage, where both systems can be applied to the same tasks. The transfer evaluation broadens the picture, but it also introduces its own

8.4. FUTURE WORK

constraints: some datasets require model-assisted filtering or annotation, VirtualHome contains only 67 questions, and no suitable external Table Setting dataset is available. The knowledge sources themselves are uneven as well. ConceptNet and the other graphs provide useful background associations, yet their coverage is incomplete and may reflect conventions that fit some households or cultures better than others.

A final boundary concerns what the benchmark measures. RoboCSKBench evaluates language-level judgements about household situations, not perception, manipulation or recovery from failed actions on a physical robot. The symbolic systems can show which facts are retrieved and which rules fire, and this makes an answer easier to inspect than a direct LLM response. Still, an inspectable derivation is not the same as a correct decision. A proof can be built from a mistaken perceptual premise, and this thesis does not measure interpretability or reliability through a dedicated user study. The conclusions should therefore be understood as claims about the evaluated datasets, models and task formats, rather than as evidence that the same architecture would automatically remain reliable in a deployed robotic system.

8.4 Future Work

Further work should first improve the interface between neural extraction and symbolic reasoning. For Pipeline 1, predicate schemas should be tested for their ability to distinguish candidate answers before rule induction, and richer hypotheses should be introduced only where the extracted features support them reliably. Pipeline 2 should represent uncertainty and missing information explicitly instead of allowing absent predicates to behave silently as false. Decision traces could then connect the original instruction, extracted facts, graph assertions and fired rules, making failures easier to locate without presenting the trace as proof that every premise is correct.

Vocabulary alignment also requires a more adaptive treatment than static synonym lists. A system should be able to normalise unfamiliar entities, estimate when knowledge-base coverage is inadequate and abstain when no grounded conclusion is available. For household tasks, user-specific preferences should override population-level defaults rather than being treated as noise.

Future work should also compare a broader range of hybrid architectures. Both pipelines in this thesis ultimately use rule-based inference through ASP or Prolog, even when knowledge graphs supply their facts. Alternative designs could reason directly over knowledge graphs, combine graph retrieval with learned link prediction, integrate differentiable logical constraints into model training, or allow neural and symbolic components to exchange intermediate results iteratively. Comparing these arrangements on the same

tasks and ablations would show whether the conditional benefits observed here are specific to rule-based pipelines or extend to other ways of combining explicit and learned knowledge.

The transfer findings motivate a pre-specified study of answer-structure alignment using several datasets for each type of shift, together with models from families beyond Mistral and an external Table Setting dataset. Construction and evaluation models should be separated where possible, and uncertainty should be reported for small samples. The most important extension is evaluation in simulation and on physical robots, where perceptual errors, action failures and environmental change can test whether the observed language-level advantages survive closed-loop execution.

8.5 Closing Remarks

The experiments do not support a universal choice between neural and symbolic reasoning. They show instead that the benefit of combining them depends on where knowledge is represented, how language is mapped to that representation and whether the rules match the decision the task requires. LLMs provide the linguistic flexibility needed to construct symbolic inputs, while knowledge bases and logic programs are most useful when they contribute explicit facts or constraints that the task can exploit. The result is a conditional but practical conclusion: neuro-symbolic reasoning is valuable when its intermediate representation is treated as part of the problem, not as an automatically correct bridge between language and action.

Bibliography

- [1] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL <https://arxiv.org/abs/2204.01691>.
- [2] E. Aker, V. Patoglu, and E. Erdem. Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics. *IFAC Proceedings Volumes*, 45(22):77–83, 2012. doi: 10.3182/20120905-3-HR-2030.00169. URL <https://doi.org/10.3182/20120905-3-HR-2030.00169>.
- [3] T. R. Besold, A. d’Avila Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kuehnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, and G. Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation, 2017. URL <https://arxiv.org/abs/1711.03902>.
- [4] D. Beßler, R. Porzel, M. Pomarlan, A. Vyas, S. Höffner, M. Beetz, R. Malaka, and J. Bateman. Foundations of the socio-physical model of activities (soma) for autonomous robotic agents, 2020. URL <https://arxiv.org/abs/2011.11972>.
- [5] M. Bień, M. Gilski, M. Maciejewska, W. Taisner, D. Wisniewski, and A. Lawrynowicz. RecipeNLG: A cooking recipes dataset for semi-structured text generation. In B. Davis, Y. Graham, J. Kelleher, and Y. Sripada, editors, *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28, Dublin, Ireland, Dec. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.inlg-1.4. URL <https://aclanthology.org/2020.inlg-1.4/>.
- [6] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi. Piqa: Reasoning about physical

- commonsense in natural language, 2019. URL <https://arxiv.org/abs/1911.11641>.
- [7] M. A. Borroto Santana, K. Gallagher, A. Ielo, I. Kareem, F. Ricca, and A. Russo. Question answering with LLMs and learning from answer sets. *Theory and Practice of Logic Programming*, pages 1–25, 2025. doi: 10.1017/S1471068425100343. URL <https://doi.org/10.1017/S1471068425100343>.
- [8] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, Dec. 2011. ISSN 0001-0782. doi: 10.1145/2043174.2043195. URL <https://doi.org/10.1145/2043174.2043195>.
- [9] A. d’Avila Garcez and L. C. Lamb. Neurosymbolic ai: The 3rd wave, 2020. URL <https://arxiv.org/abs/2012.05876>.
- [10] H.-S. Fang, H. Fang, Z. Tang, J. Liu, C. Wang, J. Wang, H. Zhu, and C. Lu. Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot, 2023. URL <https://arxiv.org/abs/2307.00595>.
- [11] I. Fang, J. Zhang, S. Tong, and C. Feng. From intention to execution: Probing the generalization boundaries of vision-language-action models, 2025. URL <https://arxiv.org/abs/2506.09930>.
- [12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Clingo = asp + control: Preliminary report, 2014. URL <https://arxiv.org/abs/1405.3694>.
- [13] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer,

BIBLIOGRAPHY

K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yearly, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsim-poukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speck-bacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Sri-vastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. San-gani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leon-hardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Han-cock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Bur-ton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tin-dal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Flo-rez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizen-stein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. Mc-

- Phie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma. The llama 3 herd of models, 2024. URL <https://arxiv.org/abs/2407.21783>.
- [14] F. Ilievski, P. Szekely, and B. Zhang. Cskg: The commonsense knowledge graph, 2021. URL <https://arxiv.org/abs/2012.11490>.
- [15] Y. Kant, A. Ramachandran, S. Yenamandra, I. Gilitschenski, D. Batra, A. Szot, and H. Agrawal. Housekeep: Tidying virtual households using commonsense reasoning, 2022. URL <https://arxiv.org/abs/2205.10712>.
- [16] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simp-

BIBLIOGRAPHY

- son, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, V. Guizilini, D. A. Herrera, M. Heo, K. Hsu, J. Hu, M. Z. Irshad, D. Jackson, C. Le, Y. Li, K. Lin, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O'Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. Droid: A large-scale in-the-wild robot manipulation dataset, 2025. URL <https://arxiv.org/abs/2403.12945>.
- [17] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, A. Kembhavi, A. Gupta, and A. Farhadi. Ai2-thor: An interactive 3d environment for visual ai, 2022. URL <https://arxiv.org/abs/1712.05474>.
- [18] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 01 1985. doi: 10.1007/BF03037383.
- [19] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. S. Bernstein, and F.-F. Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, 2016. URL <https://arxiv.org/abs/1602.07332>.
- [20] M. Law, A. Russo, and K. Broda. Inductive learning of answer set programs from noisy examples, 2018. URL <https://arxiv.org/abs/1808.08441>.
- [21] J. Lee and R. Palla. Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming. *Journal of Artificial Intelligence Research*, 43:571–620, Apr. 2012. ISSN 1076-9757. doi: 10.1613/jair.3489. URL <http://dx.doi.org/10.1613/jair.3489>.
- [22] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, W. Ai, B. Martinez, H. Yin, M. Lingelbach, M. Hwang, A. Hiranaka, S. Garlanka, A. Aydin, S. Lee, J. Sun, M. Anvari, M. Sharma, D. Bansal, S. Hunter, K.-Y. Kim, A. Lou, C. R. Matthews, I. Villa-Renteria, J. H. Tang, C. Tang, F. Xia, Y. Li, S. Savarese, H. Gweon, C. K. Liu, J. Wu, and L. Fei-Fei. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation, 2024. URL <https://arxiv.org/abs/2403.09227>.

- [23] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control, 2023. URL <https://arxiv.org/abs/2209.07753>.
- [24] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015. URL <https://arxiv.org/abs/1405.0312>.
- [25] X. Lin, Y. Wu, H. Yang, Y. Zhang, Y. Zhang, and J. Ji. CLMASP: Coupling large language models with answer set programming for robotic task planning, 2024. URL <https://arxiv.org/abs/2406.03367>.
- [26] J. Marin, A. Biswas, F. Ofli, N. Hynes, A. Salvador, Y. Aytar, I. Weber, and A. Torralba. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images, 2019. URL <https://arxiv.org/abs/1810.06553>.
- [27] J. McCarthy and P. J. Hayes. *Some philosophical problems from the standpoint of artificial intelligence*, page 26–45. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987. ISBN 0934613451.
- [28] D. Meli, M. Sridharan, and P. Fiorini. Inductive learning of answer set programs for autonomous surgical task planning. *Machine Learning*, 110(7):1739–1763, 2021. doi: 10.1007/s10994-021-06013-7. URL <https://doi.org/10.1007/s10994-021-06013-7>.
- [29] G. A. Miller. WordNet: A lexical database for English. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994. URL <https://aclanthology.org/H94-1111/>.
- [30] Mistral AI. Medium is the new large. <https://mistral.ai/it/news/mistral-medium-3/>, 2025. Accessed: 2026-06-11.
- [31] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19–20(Supplement 1):629–679, 1994. ISSN 0743-1066. doi: 10.1016/0743-1066(94)90035-3. URL [https://doi.org/10.1016/0743-1066\(94\)90035-3](https://doi.org/10.1016/0743-1066(94)90035-3).
- [32] T.-P. Nguyen, S. Razniewski, J. Romero, and G. Weikum. Refined commonsense knowledge from large-scale web contents. *IEEE Transactions on Knowledge and Data Engineering*, page 1–16, 2022. ISSN 2326-3865. doi: 10.1109/tkde.2022.3206505. URL <http://dx.doi.org/10.1109/TKDE.2022.3206505>.

BIBLIOGRAPHY

- [33] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O’Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner,

- C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. Gpt-4 technical report, 2024. URL <https://arxiv.org/abs/2303.08774>.
- [34] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu, and A. Miller. Language models as knowledge bases? In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, Nov. 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL <https://aclanthology.org/D19-1250/>.
- [35] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs, 2018. URL <https://arxiv.org/abs/1806.07011>.
- [36] E. Saccon, A. Tikna, D. De Martini, E. Lamon, L. Palopoli, and M. Roveri. When Prolog meets generative models: A new approach for managing knowledge and planning in robotic applications. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 17065–17071, 2024. doi: 10.1109/ICRA57147.2024.10610800. URL <https://doi.org/10.1109/ICRA57147.2024.10610800>.
- [37] E. Şahin, M. Çakmak, M. R. Doğar, E. Uğur, and G. Üçoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007. doi: 10.1177/1059712307084689.
- [38] M. Shanahan. *The Event Calculus Explained*, volume 1600, pages 409–430. 05 2001. ISBN 978-3-540-66428-4. doi: 10.1007/3-540-48317-9_17.
- [39] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks, 2020. URL <https://arxiv.org/abs/1912.01734>.
- [40] R. Speer, J. Chin, and C. Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge, 2018. URL <https://arxiv.org/abs/1612.03975>.
- [41] G. Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret, P. Liu, P. Tafti, A. Friesen,

BIBLIOGRAPHY

- M. Casbon, S. Ramos, R. Kumar, C. L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J.-B. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C. A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozińska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucińska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J. P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. yeong Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. Lago, L. McNealus, L. B. Soares, L. Kilpatrick, L. Dixon, L. Martins, M. Reid, M. Singh, M. Iverson, M. Görner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. Bardoliwalla, N. Devanathan, N. Dumai, N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. Jin, P. Georgiev, P. Culliton, P. Kuppala, R. Comanescu, R. Merhej, R. Jana, R. A. Rokni, R. Agarwal, R. Mullins, S. Saadat, S. M. Carthy, S. Cogan, S. Perrin, S. M. R. Arnold, S. Krause, S. Dai, S. Garg, S. Sheth, S. Ronstrom, S. Chan, T. Jordan, T. Yu, T. Eccles, T. Hennigan, T. Kocisky, T. Doshi, V. Jain, V. Yadav, V. Meshram, V. Dharmadhikari, W. Barkley, W. Wei, W. Ye, W. Han, W. Kwon, X. Xu, Z. Shen, Z. Gong, Z. Wei, V. Cotruta, P. Kirk, A. Rao, M. Giang, L. Peran, T. Warkentin, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, D. Sculley, J. Banks, A. Dragan, S. Petrov, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, S. Borgeaud, N. Fiedel, A. Joulin, K. Kenealy, R. Dadashi, and A. Andreev. Gemma 2: Improving open language models at a practical size, 2024. URL <https://arxiv.org/abs/2408.00118>.
- [42] M. Tenorth and M. Beetz. Knowrob — knowledge processing for autonomous personal robots. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266, 2009. doi: 10.1109/IROS.2009.5354602.
- [43] J.-P. Töberg, A.-C. Ngonga Ngomo, M. Beetz, and P. Cimiano. Commonsense knowledge in cognitive robotics: a systematic literature review. *Frontiers in Robotics and AI*, 11:1328934, 2024. doi: 10.3389/frobt.2024.1328934. URL <https://doi.org/10.3389/frobt.2024.1328934>.

BIBLIOGRAPHY

- [44] J.-P. Töberg, S. Kenneweg, and P. Cimiano. Robocskbench: Benchmarking embodied commonsense capabilities of large language models. In *2025 22nd International Conference on Ubiquitous Robots (UR)*, pages 199–206, College Station, Texas, USA, 2025. doi: 10.1109/UR65550.2025.11078036.
- [45] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. Swi-prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. doi: 10.1017/S1471068411000494. URL <https://doi.org/10.1017/S1471068411000494>.
- [46] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. Tidybot: personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, Nov. 2023. ISSN 1573-7527. doi: 10.1007/s10514-023-10139-z. URL <http://dx.doi.org/10.1007/s10514-023-10139-z>.

Appendix A

Appendix

A.1 Extracted Domestic Subgraph Format

The following is a representative snippet of the JSON output structure generated after the graph extraction and filtering phase (specifically from `object_locations_cskg.json`). It maps each discovered domestic object to its spatial locations, source provenance, hierarchical inheritance context, and originating seed rooms:

```
{
  "blender": {
    "locations": [
      "kitchen"
    ],
    "sources": {
      "kitchen": [
        "csvg:cn"
      ]
    },
    "inherited_from": [],
    "discovered_via_rooms": [
      "kitchen"
    ]
  },
  "coffee mug": {
    "locations": [
      "cupboard",
      "kitchen",
      "shelf"
    ]
  }
}
```

```

],
"sources": {
  "kitchen": [
    "csvg:cn"
  ],
  "cupboard": [
    "csvg:cn"
  ],
  "shelf": [
    "csvg:cn"
  ]
},
"inherited_from": [],
"discovered_via_rooms": [
  "kitchen"
]
}
}

```

A.2 Extracted Tool Knowledge Base Format

The following is a representative snippet of the JSON output structure generated after the CSKG affordance extraction and inheritance pipeline for the Tool Usage task (specifically from `tool_affordances_csvg.json`). It maps discovered household objects to canonical affordance lists, source provenance mappings, inheritance history, lookup keys, seed discovery rooms, and direct raw graph phrases:

```

{
  "furnace": {
    "canonical": [
      "heat"
    ],
    "sources": {
      "heat": [
        "csvg:cn"
      ]
    },
    "inherited_from": [],

```

A.2. EXTRACTED TOOL KNOWLEDGE BASE FORMAT

```
"lookup_key": "furnace",
"discovered_via_rooms": [
  "basement",
  "cellar",
  "bedroom",
  "closet"
],
"direct_raw": [
  "heat building",
  "heat home",
  "heat house",
  "warm home",
  "heating things such as pottery"
]
},
"vacuum": {
  "canonical": [
    "stick",
    "clean"
  ],
  "sources": {
    "stick": [
      "csvg:vg"
    ],
    "clean": [
      "csvg:cn"
    ]
  }
},
"inherited_from": [],
"lookup_key": "vacuum",
"discovered_via_rooms": [
  "closet",
  "garage",
  "bedroom",
  "kitchen",
  "pantry"
],
```

```

    "direct_raw": [
        "stick",
        "clean carpet",
        "clean floor"
    ]
}
}

```

A.3 Pipeline 1 Large Language Model Prompts

This section documents the structured prompts used across the different phases of Pipeline 1.

A.3.1 Semantic Object Filtering Prompt (Llama 3.1:8b)

The following system prompt is used during the offline phase to clean the extracted CSKG subgraphs, filtering out non-domestic or abstract concepts:

You are a filter for a domestic robot dataset. Your job is to decide if a concept can realistically be present inside or around a home.

KEEP an item if it belongs to ANY of these categories:

1. Furniture and fixtures: chair, table, bed, sofa, lamp, shelf, bookcase, wardrobe, cabinet, door, window, staircase, wall, ceiling, floor, roof...
2. Appliances and electronics: fridge, washing machine, oven, TV...
3. Kitchenware and tableware: knife, fork, spoon, pot, pan, bowl, plate, cup...
4. Food and drinks: bread, wine, beer, milk, water, juice, tuna...
5. Cleaning and maintenance supplies: brush, mop, broom, vacuum, detergent...
6. Office and craft supplies: pencil, pen, stapler, staples, tape, scissors...
7. Clothing and accessories: shirt, dress, socks, jeans, suit, belt, shoes...
8. Materials found in a home: wood, cotton, velvet, silk, nylon, steel, glass...
9. Structural elements: wall, ceiling, floor, stairs, railing, foundation...
10. Personal care items: shampoo, toothbrush, razor, comb, mirror, lotion...
11. Household pets and plants: cat, dog, rabbit, hamster, fish, cactus...
12. Waste and storage: garbage, trash, dust, box, bag, container, jar, bottle...

DISCARD an item ONLY if it is one of these:

- A person or human role: person, student, chef, actor, doctor, neighbor...
- A purely abstract concept with no physical form: freedom, love, idea, time...
- A non-domestic vehicle: airplane, car, motorcycle, truck, submarine, bus
- A non-domestic outdoor place: city, street, ocean, lake, island, park...
- A wild or farm animal not kept as a pet: bear, wolf, whale, dolphin...

A.3. PIPELINE 1 LARGE LANGUAGE MODEL PROMPTS

- A non-physical media/entertainment concept: music, TV show, movie...
- A clearly dangerous/illegal item not normally at home: dynamite, bomb...

When in doubt, KEEP the item. It is better to keep something borderline than to discard something that could be in a home.

Return ONLY a JSON object with one key:

```
{"keep": ["item1", "item2", ...]}
```

A.3.2 SOMA Ontology Annotation Prompt (Llama 3.1:8b)

The system prompt used to perform semantic annotation of domestic objects with SOMA ontology predicates is structured as follows:

You are a semantic annotator for a domestic robot knowledge base.
Given an object and its ConceptNet associations, assign SOMA ontology labels.
Use YOUR OWN KNOWLEDGE as primary truth. ConceptNet data are noisy - ignore them if wrong.

Return a JSON with EXACTLY three keys. Use ONLY values from these lists:

"hasPhysicalQuality" - pick from:

RequiresCooling, RequiresFreezing, Perishable, Heated, Liquid, Granular, Gas_Aerosol, Heavy, Lightweight, Fragile, Rigid, Soft_Deformable, Sharp, Electronic, Toxic_Hazardous, Flammable, Washable, HasBlade, HasPointedTip, IsRotary, HasAbrasiveSurface, HasContainer, HasHeatElement, IsFlexible, HasTeeth

"hasRole" - pick from:

ConsumableRole, UtensilRole, CookwareRole, ToolRole, CleaningToolRole, ApplianceRole, StorageContainerRole, FurnitureRole, BeddingRole, LightingRole, ClothingRole, EntertainmentRole, MusicalInstrumentRole, DecorationRole, DocumentRole, WasteRole, SafetyEquipmentRole

"affordsTask" - pick from:

FoodPreparationTask, EatingDrinkingTask, MaintenanceTask, CleaningTask, WasteDisposalTask, HygieneTask, DressingTask, StorageTask, SleepingTask, WorkStudyTask, LeisureTask, GardeningTask

COMMON MISTAKES TO AVOID:

- Fridge/freezer/air conditioner -> NOT Heated! They COOL things.
- Chocolate/butter/fruit -> NOT Liquid! They are solid at room temperature.
- Cookies/fruit -> NOT Granular! They are solid pieces.
- Teddy bear/stuffed animal -> EntertainmentRole, NOT FurnitureRole.
- Bandaid/plaster -> SafetyEquipmentRole + HygieneTask, NOT Liquid or ConsumableRole.

- Hammer/vice -> MaintenanceTask, NOT FoodPreparationTask.
- Umbrella/hairbrush -> ToolRole, NOT UtensilRole.
- Egg shells -> WasteRole, NOT ConsumableRole.
- Saw -> HasBlade AND HasTeeth. Knife -> HasBlade but NOT HasTeeth.
- Drill -> IsRotary AND HasPointedTip.
- Bowl/cup -> HasContainer. Knife -> NOT HasContainer.
- Rope/hose -> IsFlexible. Hammer -> NOT IsFlexible.

EXAMPLES:

```
knife:      {"hasPhysicalQuality":["Sharp","Rigid"],
             "hasRole":["UtensilRole"], "affordsTask":["FoodPreparationTask"]}
shampoo:    {"hasPhysicalQuality":["Liquid"],
             "hasRole":["CleaningToolRole"], "affordsTask":["HygieneTask"]}
fridge:     {"hasPhysicalQuality":["Electronic","Heavy"],
             "hasRole":["ApplianceRole"], "affordsTask":["StorageTask"]}
teddy bear: {"hasPhysicalQuality":["Soft_Deformable","Lightweight"],
             "hasRole":["EntertainmentRole"], "affordsTask":["LeisureTask"]}
bandaid:    {"hasPhysicalQuality":["Lightweight"],
             "hasRole":["SafetyEquipmentRole"], "affordsTask":["HygieneTask"]}
chocolate: {"hasPhysicalQuality":["Perishable"],
             "hasRole":["ConsumableRole"], "affordsTask":["EatingDrinkingTask"]}
hammer:     {"hasPhysicalQuality":["Heavy","Rigid"],
             "hasRole":["ToolRole"], "affordsTask":["MaintenanceTask"]}
saw:        {"hasPhysicalQuality":["Heavy","Rigid","HasBlade","HasTeeth"],
             "hasRole":["ToolRole"], "affordsTask":["MaintenanceTask"]}
drill:      {"hasPhysicalQuality":["Heavy","Rigid","IsRotary","HasPointedTip"],
             "hasRole":["ToolRole"], "affordsTask":["MaintenanceTask"]}
blanket:    {"hasPhysicalQuality":["Soft_Deformable","Washable"],
             "hasRole":["BeddingRole"], "affordsTask":["SleepingTask"]}
```

Return ONLY the JSON. No explanation, no markdown, no extra text.

A.3.3 LLM-Guided Rule Induction System Prompt (Mistral Medium)

The system prompt used to guide the rule induction generator (Mistral Medium) during the iterative hypothesis generation loop is structured as follows:

You are an expert in Answer Set Programming (ASP).
 Generate ONLY well-formed ASP rules.
 PRIORITY: Cover the FOCUS OBJECT. It is acceptable to also cover other positives.
 It is acceptable if false positives have the SAME properties as true positives
 (they may just be mislabeled or assigned to different locations).
 Do not output explanations, markdown, or meta-commentary.
 Output EXACTLY ONE rule per response.

A.4. PIPELINE 2 LARGE LANGUAGE MODEL PROMPTS

CRITICAL INSTRUCTION - MINIMAL RULES ONLY:

- You MUST generate the SHORTEST POSSIBLE rule.
- Use the absolute MINIMUM number of conditions necessary to distinguish the focus object from negatives.
- DO NOT just copy all properties of the focus object. Choose 1 or 2 most important discriminative features.

CRITICAL SYNTAX RULES:

- Do NOT use semicolons (;) at the top level of the rule body. Disjunction at the top level creates invalid ASP. Use commas only.
- Example of INVALID rule:
goesIn(X, loc) :- obj(X), hasRole(X, roleA) ; hasRole(X, roleB).
- Example of VALID rule:
goesIn(X, loc) :- obj(X), hasRole(X, roleA), affordsTask(X, taskB).
- Example of VALID rule:
goesIn(X, loc) :- obj(X), hasPhysicalQuality(X, requiresCooling).
- Write the MOST GENERAL rule possible. Use the MINIMUM number of conditions necessary to distinguish the positives from the negatives. Do not just copy all properties of the focus object.

VALID ONTOLOGY VALUES - you MUST only use these exact strings, nothing else:

Qualities: [list of valid SOMA physical qualities]

Roles: [list of valid SOMA roles]

Tasks: [list of valid SOMA tasks]

Using any value NOT in these lists will cause the rule to cover nothing and be rejected.

A.4 Pipeline 2 Large Language Model Prompts

This section documents the LLM extraction prompts used in the neural phases of Pipeline 2.

A.4.1 Location Type Extraction Prompt (Tidy Up)

The system prompt used by Mistral for mapping household objects to canonical locations is structured as follows:

You are a commonsense knowledge assistant for a household robot.

Given a household object, identify the single most appropriate canonical location TYPE where this object is typically stored or placed in a home.

Choose EXACTLY ONE canonical location from the following list:

bathroom, bed, bedroom, cabinet, closet, dining_room, drawer, fridge, garage,

APPENDIX A. APPENDIX

garden, hallway, home_office, kitchen, laundry_room, library, living_room,
music_room, pantry, playroom, shelf, sink, storage_room, table, trash

Rules:

- Output ONLY the single canonical location key exactly as it appears above.
- Choose the location where the object is MOST COMMONLY stored.
- If the object is food or a consumable, prefer "kitchen", "fridge", or "pantry".
- If the object is clothing or accessories, prefer "bedroom", "closet",
or "laundry_room".
- If the object is a tool/appliance, prefer "kitchen", "garage", "home_office",
or "storage_room".
- If the object is a toy/game, prefer "bedroom", "playroom", or "living_room".
- If the object is a book, prefer "library", "home_office", or "shelf".
- Outdoor equipment, vehicles, or garden tools -> "garage" or "garden".
- Cleaning supplies -> "laundry_room" or "bathroom".
- Medicine or personal care items -> "bathroom" or "cabinet".
- Do NOT output explanations or anything other than the single canonical key.

First provide a 1-sentence "reasoning", then the "location".

Respond ONLY with a JSON dictionary:

```
{"reasoning": "<1-sentence explanation>", "location": "<canonical location key>"}
```

EXAMPLES:

Object: apple

```
{"reasoning": "Fruit is typically stored in the kitchen or fridge.",  
  "location": "kitchen"}
```

Object: jacket

```
{"reasoning": "Clothing is stored in the bedroom closet or wardrobe.",  
  "location": "closet"}
```

Object: hammer

```
{"reasoning": "Tools are typically stored in the garage.", "location": "garage"}
```

Object: novel

```
{"reasoning": "Books are stored on a shelf or in a library.",  
  "location": "library"}
```

Object: shampoo

```
{"reasoning": "Personal care items are stored in the bathroom.",  
  "location": "bathroom"}
```

Object: lawnmower

```
{"reasoning": "Garden equipment is kept in the garage.", "location": "garage"}
```

A.4. PIPELINE 2 LARGE LANGUAGE MODEL PROMPTS

A.4.2 Affordance Extraction Prompt (Tool Usage)

The system prompt used by Mistral for mapping natural language task descriptions to canonical physical affordances is structured as follows:

You are a commonsense knowledge assistant for a household robot.
Given a household task description, you must identify which single affordance (physical capability) an object must have to be useful for this task.

Choose EXACTLY ONE affordance from the following canonical list:
absorb, be heated, beautify, bend, bore, break, carry, carve, change, chop, clean, comfort, compress, contain, control, cool, cover, crack, crumble, crunch, cube, cut, decorate, dice, dig, display, dispose, drill, dry, eat, engrave, entertain, exercise, file, fix, flex, grasp, grind, hammer, handle, heat, hold, illuminate, increase, install, lift, mark, melt, mix, operate, peel, pick, pierce, plow, plug, poke, polish, pour, press, read, repair, roll, rub, saw, scoop, screw, separate, shape, skin, slit, smoothen, solder, spread, staple, stick, store, strip, support, tamp, time, unclog, wash, wipe, wrap, wrench, write

Rules:

- Output ONLY the single affordance word/phrase, exactly as it appears above.
- If the task involves placing a container on heat (boiling, reheating, cooking food in a pot/pan), output "be heated".
- If the task involves melting something (wax, butter, chocolate, ice) using a tool that PROVIDES heat, output "heat".
- If the task involves chopping food into pieces, output "chop". Reserve "cut" for slicing/trimming tasks (e.g., cutting paper, cutting fabric).
- If the task involves breaking an object apart, output "break". If cracking eggs or nuts, output "crack". If crumbling, output "crumble". If crunching hard food, output "crunch". If dicing into small cubes, output "dice".
- If the task involves boring or making a hole for screws (assembling furniture, hanging pictures, installing fixtures), output "bore". Output "screw" only for driving/tightening screws explicitly.
- If the task involves drilling a hole with a powered drill, output "drill".
- If the task involves hammering a nail, output "hammer".
- If the task involves cleaning, dusting, or removing dirt, output "clean". Use "wipe" only for wiping up liquid spills. Use "wash" for rinsing.
- If the task involves mixing, blending, or stirring ingredients, output "mix".
- If the task involves pressing a button or operating gadgets, output "press".
- If the task involves piercing, puncturing, or creating a hole, output "pierce".
- If the task involves physically lifting or moving objects, output "lift". If gripping/grasping a specific item, output "grasp".
- Do NOT output explanations or anything other than the single affordance.

First provide a 1-sentence "reasoning", then the "affordance".

Respond ONLY with a JSON dictionary:

```
{"reasoning": "<1-sentence explanation>", "affordance": "<canonical affordance>"}
```

EXAMPLES:

Task: cleaning up spills

```
{"reasoning": "A sponge-like object that absorbs liquid is needed.",
  "affordance": "absorb"}
```

Task: reheating coffee

```
{"reasoning": "The object must be heatable on a stove or similar.",
  "affordance": "be heated"}
```

Task: cutting meat

```
{"reasoning": "A sharp cutting tool is required.", "affordance": "cut"}
```

Task: hanging pictures

```
{"reasoning": "A tool to bore a hole into the wall is needed.",
  "affordance": "bore"}
```

Task: shaping wire

```
{"reasoning": "A tool that can bend metal is required.", "affordance": "bend"}
```

Task: installing shelves

```
{"reasoning": "A boring tool is needed to make pilot holes for screws.",
  "affordance": "bore"}
```

Task: assembling furniture

```
{"reasoning": "Pilot holes must be bored before inserting screws.",
  "affordance": "bore"}
```

Task: boiling water

```
{"reasoning": "A container that can be placed on heat is needed.",
  "affordance": "be heated"}
```

A.4.3 Robot Capability Extraction Prompt (Meta-Reasoning)

The system prompt used by Mistral for identifying physical task requirements for robot motion planning is structured as follows:

You are an expert robot motion planner.

Your job is to analyze a natural language household task and answer 6 specific Yes/No questions about its physical requirements.

First, provide a brief 1-sentence rationale in the "reasoning" key. Then, output

A.4. PIPELINE 2 LARGE LANGUAGE MODEL PROMPTS

the 6 booleans.

Focus strictly on the MINIMUM physical actions required. Think like a roboticist:

1. "needs_grasping": Does this task strictly require lifting an object completely into the air? Output FALSE if the task can theoretically be accomplished by simply pushing, sweeping, or sliding the object with a blunt arm. If the task includes the word "arrange", needs_grasping is FALSE.
2. "needs_precision": Does this task involve highly constrained, multi-axis 7-DoF rotations (like inserting a key, threading a needle)? Output FALSE for general manipulation, arranging, pushing, or broad adjustments.
3. "needs_mobility": Does the robot strictly need a mobile base? Output FALSE unless the task absolutely cannot be done from a single position. Assume the robot is ALWAYS already positioned optimally within arm's reach of ALL objects.
4. "needs_bimanual": Does the task ABSOLUTELY require two arms working simultaneously where NO passive support is possible? Output FALSE for cutting, slicing, or chopping (the surface stabilizes the food). Output FALSE for folding (can be done sequentially). Only output TRUE for tasks like tearing an object apart, where two opposing forces must be applied simultaneously.
5. "needs_contact": Does the robot need to physically touch anything at all?
6. "needs_rigid_grip": Is the object heavy, resistive, or requiring a firm rigid grip? Output FALSE if the task can be done by pushing (e.g. arranging a sofa).

Respond ONLY with a valid JSON dictionary matching exactly the keys above.

EXAMPLES:

Task: arrange shoes

```
{
  "reasoning": "Arranging shoes can be done by pushing or sliding them.",
  "needs_grasping": false,
  "needs_precision": false,
  "needs_mobility": false,
  "needs_bimanual": false,
  "needs_contact": true,
  "needs_rigid_grip": false
}
```

Task: adjust the desk lamp

```
{
  "reasoning": "Adjusting a lamp requires physical contact to nudge it.",
  "needs_grasping": false,
  "needs_precision": false,
  "needs_mobility": false,
  "needs_bimanual": false,
  "needs_contact": true,
  "needs_rigid_grip": false
}
```

```
}
```

Task: bring the bowl to the counter

```
{
  "reasoning": "Bringing an object requires the arm to reach.",
  "needs_grasping": true,
  "needs_precision": false,
  "needs_mobility": false,
  "needs_bimanual": false,
  "needs_contact": true,
  "needs_rigid_grip": false
}
```

Task: slice a tomato on a cutting board

```
{
  "reasoning": "The cutting board stabilizes the tomato passively.",
  "needs_grasping": true,
  "needs_precision": false,
  "needs_mobility": false,
  "needs_bimanual": false,
  "needs_contact": true,
  "needs_rigid_grip": true
}
```

Task: get the pen from the table and go turn on the floor lamp

```
{
  "reasoning": "Robot is assumed to be within arm's reach of all objects.",
  "needs_grasping": true,
  "needs_precision": false,
  "needs_mobility": false,
  "needs_bimanual": false,
  "needs_contact": true,
  "needs_rigid_grip": false
}
```

A.4.4 Meal Property Extraction Prompt (Table Setting)

The system prompt used by Mistral for identifying semantic dining features of recipes is structured as follows:

You are an expert chef and culinary analyst.

Your job is to analyze a given meal/recipe and extract semantic physical properties about it to help determine the proper cutlery and plate.

Extract the following boolean (true/false) properties for the meal:

A.4. PIPELINE 2 LARGE LANGUAGE MODEL PROMPTS

1. "is_soup_or_broth": Is the meal predominantly liquid (e.g. soup, stew, broth)?
2. "is_handheld_food": Is it usually eaten by grabbing it with bare hands?
3. "needs_cutting": Does the dish contain large, solid chunks of meat/food?
4. "is_asian_cuisine": Is this a traditional East/Southeast Asian dish where chopsticks are commonly appropriate? If South/Central Asian, set false.
5. "is_long_noodles": Does it feature long strands of pasta or noodles?
6. "has_sauce_or_small_pieces": Is it composed of loose small pieces that are commonly scooped OR a thick curry-like dish served in a bowl?
7. "is_dessert": Is this a dessert or sweet baked good?
8. "is_skewer_food": Is this food typically served and eaten on a skewer/stick?
9. "is_tongs_food": Is this food strictly requiring tongs to eat?

First provide a 1-sentence "reasoning", then output the boolean constraints.

Respond ONLY with a valid JSON dictionary. Do NOT add any text outside the JSON.

EXAMPLES:

Meal: Beef Barbecue Casserole

```
{
  "reasoning": "This is a baked casserole served plated.",
  "is_soup_or_broth": false,
  "is_handheld_food": false,
  "needs_cutting": false,
  "is_asian_cuisine": false,
  "is_long_noodles": false,
  "has_sauce_or_small_pieces": false,
  "is_dessert": false,
  "is_skewer_food": false,
  "is_tongs_food": false
}
```

Meal: Hamburgers

```
{
  "reasoning": "Hamburgers are classic handheld foods eaten without utensils.",
  "is_soup_or_broth": false,
  "is_handheld_food": true,
  "needs_cutting": false,
  "is_asian_cuisine": false,
  "is_long_noodles": false,
  "has_sauce_or_small_pieces": false,
  "is_dessert": false,
  "is_skewer_food": false,
  "is_tongs_food": false
}
```

Meal: Banh Mi (Vietnamese Sandwich)

```
{
  "reasoning": "This is a handheld sandwich; not chopsticks-appropriate.",
  "is_soup_or_broth": false,
  "is_handheld_food": true,
  "needs_cutting": false,
  "is_asian_cuisine": false,
  "is_long_noodles": false,
  "has_sauce_or_small_pieces": false,
  "is_dessert": false,
  "is_skewer_food": false,
  "is_tongs_food": false
}
```

Meal: Steak With Peppercorn Sauce

```
{
  "reasoning": "This is a plated Western meat dish that requires cutting.",
  "is_soup_or_broth": false,
  "is_handheld_food": false,
  "needs_cutting": true,
  "is_asian_cuisine": false,
  "is_long_noodles": false,
  "has_sauce_or_small_pieces": false,
  "is_dessert": false,
  "is_skewer_food": false,
  "is_tongs_food": false
}
```

A.4.5 Recipe Step Property Extraction Prompt (Procedural Knowledge)

The system prompt used by Mistral for identifying recipe step attributes is structured as follows:

You are an expert chef and cooking procedure analyst.

Your job is to analyze two steps from a cooking recipe and extract semantic properties that reveal their temporal order.

Given the recipe title and two steps (step_a and step_b), extract the following properties for EACH step:

1. "cooking_phase": Which phase of cooking does this step belong to? Choose ONE:
 - "preheating" - Boiling initial water, heating oven before anything else
 - "prep" - Washing, peeling, chopping, dicing, mincing raw ingredients

A.4. PIPELINE 2 LARGE LANGUAGE MODEL PROMPTS

- "mixing" - Combining raw ingredients together (BEFORE cooking)
 - "marinating" - Soaking, brining, marinating
 - "heating" - Heating oil in a pan, melting butter (getting pan hot)
 - "cooking" - Main cooking process: bake, fry, sauté, simmer, boil
 - "assembly" - Layering, stuffing, rolling cooked components together
 - "resting" - Letting dough rise, letting meat rest, cooling slightly
 - "cooling" - Active cooling: chill in fridge, freeze, cool completely
 - "finishing" - Final seasoning, glazing, garnishing, drizzling
 - "serving" - Plating, serving, presenting the dish
2. "requires_heat": Does this step involve applying heat? (true/false)
3. "is_final_step": Is this one of the very last things done? (true/false)
4. "transforms_state": Does this step change the physical state? (true/false)
5. "depends_on_other": Does this step require the OTHER step completed? (true/false)
6. "belongs_to_recipe": Does this step belong to this specific recipe? (true/false)
- "Soak Basmati rice" belongs to "Beetroot Rice" -> true
 - "Broil swordfish" does NOT belong to "Green Beans With Onion" -> false
 - When in doubt, output true.

Respond ONLY with a valid JSON dictionary in this exact format:

```
{
  "reasoning": "<1-2 sentence explanation of the temporal relationship>",
  "step_a": {
    "cooking_phase": "<phase>",
    "requires_heat": <bool>,
    "is_final_step": <bool>,
    "transforms_state": <bool>,
    "depends_on_other": <bool>,
    "belongs_to_recipe": <bool>
  },
  "step_b": {
    "cooking_phase": "<phase>",
    "requires_heat": <bool>,
    "is_final_step": <bool>,
    "transforms_state": <bool>,
    "depends_on_other": <bool>,
    "belongs_to_recipe": <bool>
  }
}
```

Do NOT add any text outside the JSON.

EXAMPLES:

Recipe: World's Best Mac and Cheese

Step A: Cook pasta less than directions

Step B: Bake pasta with cheese topping

```
{
  "reasoning": "Pasta must be cooked first before it can be baked.",
  "step_a": {
    "cooking_phase": "cooking",
    "requires_heat": true,
    "is_final_step": false,
    "transforms_state": true,
    "depends_on_other": false,
    "belongs_to_recipe": true
  },
  "step_b": {
    "cooking_phase": "cooking",
    "requires_heat": true,
    "is_final_step": false,
    "transforms_state": true,
    "depends_on_other": true,
    "belongs_to_recipe": true
  }
}
```

A.5 Pipeline 2 Prolog Programs

This section provides the full source listings for the SWI-Prolog reasoning programs used in Pipeline 2.

A.5.1 Tidy Up Reasoner

The Prolog program for the Tidy Up candidate selection is structured as follows:

```
% Dynamic declarations for assertions from Python
:- dynamic candidate/1.
:- dynamic has_location/2.
:- dynamic kb_confirmed/2.
:- dynamic required_canonical/1.

% Collect all candidates whose mapped canonical matches the required one.
matching_candidates(Matches) :-
  required_canonical(Canon),
  findall(C, (candidate(C), has_location(C, Canon)), Matches).
```

A.5. PIPELINE 2 PROLOG PROGRAMS

```
% Collect only the KB-confirmed subset.
kb_confirmed_candidates(Canon, KBMatches) :-
    findall(C,
        (candidate(C), has_location(C, Canon), kb_confirmed(C, Canon)),
        KBMatches).

% Select the best single candidate:
select_location :-
    required_canonical(Canon),
    kb_confirmed_candidates(Canon, KBMatches),
    length(KBMatches, 1),
    KBMatches = [Best],
    !,
    write('SELECTED='), write(Best), nl,
    write('CONFIDENCE=kb_unique'), nl.

select_location :-
    matching_candidates(Matches),
    length(Matches, 1),
    Matches = [Best],
    !,
    write('SELECTED='), write(Best), nl,
    write('CONFIDENCE=unique'), nl.

select_location :-
    matching_candidates([]),
    !,
    write('SELECTED=none'), nl,
    write('CONFIDENCE=no_match'), nl.

select_location :-
    matching_candidates(Matches),
    length(Matches, N), N > 1,
    write('SELECTED=ambiguous'), nl,
    write('AMBIGUOUS_SET='), write(Matches), nl,
    required_canonical(Canon),
    kb_confirmed_candidates(Canon, KBMatches),
    write('KB_CONFIRMED_SET='), write(KBMatches), nl,
    write('CONFIDENCE=ambiguous'), nl.

% Entry point
run :-
    select_location,
    halt.
```

A.5.2 Tool Usage Reasoner

The Prolog program for the Tool Usage candidate selection is structured as follows:

```
% Dynamic declarations for assertions from Python
:- dynamic candidate/1.
:- dynamic has_affordance/2.
:- dynamic cn_confirmed/2.
:- dynamic required_affordance/1.

% Collect all candidates with the required affordance into a list.
matching_candidates(Matches) :-
    required_affordance(Aff),
    findall(T, (candidate(T), has_affordance(T, Aff)), Matches).

% Collect only the CN-confirmed subset of matches.
cn_confirmed_candidates(Aff, CNMatches) :-
    findall(T,
        (candidate(T), has_affordance(T, Aff), cn_confirmed(T, Aff)),
        CNMatches).

% Select the best single tool:
select_tool :-
    required_affordance(Aff),
    cn_confirmed_candidates(Aff, CNMatches),
    length(CNMatches, 1),
    CNMatches = [Best],
    !,
    write('SELECTED='), write(Best), nl,
    write('CONFIDENCE=cn_unique'), nl.

select_tool :-
    matching_candidates(Matches),
    length(Matches, 1),
    Matches = [Best],
    !,
    write('SELECTED='), write(Best), nl,
    write('CONFIDENCE=unique'), nl.

select_tool :-
    matching_candidates([]),
    !,
    write('SELECTED=none'), nl,
    write('CONFIDENCE=no_match'), nl.

select_tool :-
```

A.5. PIPELINE 2 PROLOG PROGRAMS

```
    matching_candidates(Matches),
    length(Matches, N), N > 1,
    write('SELECTED=ambiguous'), nl,
    write('AMBIGUOUS_SET='), write(Matches), nl,
    required_affordance(Aff),
    cn_confirmed_candidates(Aff, CNMatches),
    write('CN_CONFIRMED_SET='), write(CNMatches), nl,
    write('CONFIDENCE=ambiguous'), nl.

% Entry point
run :-
    select_tool,
    halt.
```

A.5.3 Meta-Reasoning Capability Matcher

The Prolog program for reasoning about robot capability compatibility is structured as follows:

```
:- dynamic task_needs_grasping/1.
:- dynamic task_needs_precision/1.
:- dynamic task_needs_mobility/1.
:- dynamic task_needs_bimanual/1.
:- dynamic task_needs_contact/1.
:- dynamic task_needs_rigid_grip/1.

:- dynamic robot_mobile/1, robot_arms/1, robot_dof/1.
:- dynamic robot_gripper_type/1, robot_rigid/1.

% --- Required Arms ---
required_arms(2) :-
    task_needs_bimanual(true).
required_arms(1) :-
    \+ task_needs_bimanual(true),
    (task_needs_contact(true) ; task_needs_grasping(true)).
required_arms(0) :-
    \+ task_needs_bimanual(true),
    \+ task_needs_contact(true),
    \+ task_needs_grasping(true).

% --- Required DoF ---
required_dof(0) :-
    required_arms(0).
required_dof(6) :-
    \+ required_arms(0),
```

```

    task_needs_precision(true).
required_dof(4) :-
    \+ required_arms(0),
    \+ task_needs_precision(true).

% --- Required Gripper ---
required_gripper(true) :-
    task_needs_grasping(true).
required_gripper(false) :-
    \+ task_needs_grasping(true).

% --- Required Mobility ---
required_mobile(true) :-
    task_needs_mobility(true).
required_mobile(false) :-
    \+ task_needs_mobility(true).

% --- Required Rigid Grip ---
required_rigid(true) :-
    task_needs_rigid_grip(true).
required_rigid(false) :-
    \+ task_needs_rigid_grip(true).

% --- relaxed execution check ---
can_execute :-
    check_mobile,
    check_arms,
    check_dof,
    check_gripper,
    check_rigid,
    check_min_capability.

% --- strict execution check ---
can_execute_strict :-
    check_mobile,
    check_arms,
    check_dof_strict,
    check_gripper_strict,
    check_rigid_strict,
    check_min_capability.

% --- individual component checks ---
check_mobile :-
    required_mobile(true),
    robot_mobile(true).

```

A.5. PIPELINE 2 PROLOG PROGRAMS

```
check_mobile :-
    required_mobile(false).

check_arms :-
    required_arms(R),
    robot_arms(A),
    A >= R.

check_dof :-
    required_dof(R),
    robot_dof(D),
    D >= R.

check_gripper :-
    required_gripper(true),
    robot_arms(A),
    A >= 1.
check_gripper :-
    required_gripper(false).

check_rigid :-
    required_rigid(true),
    robot_arms(A),
    A >= 1.
check_rigid :-
    required_rigid(false).

check_min_capability :-
    task_needs_contact(false),
    \+ task_needs_grasping(true),
    robot_arms(A), A >= 1,
    !,
    robot_dof(D), D >= 3.
check_min_capability.

check_dof_strict :-
    required_arms(0),
    robot_dof(_).
check_dof_strict :-
    \+ required_arms(0),
    task_needs_precision(true),
    robot_dof(D),
    D >= 7.
check_dof_strict :-
    \+ required_arms(0),
```

```

\+ task_needs_precision(true),
robot_dof(D),
D >= 6.

check_gripper_strict :-
    required_gripper(true),
    robot_gripper_type(two_finger).
check_gripper_strict :-
    required_gripper(false).

check_rigid_strict :-
    required_rigid(true),
    robot_rigid(true).
check_rigid_strict :-
    required_rigid(false).

% --- Deficiency backtracing ---
explain_deficiencies :-
    findall(D, check_single_deficiency(D), Deficiencies),
    (   Deficiencies = []
    ->  write('no_deficiencies')
    ;   write(Deficiencies)
    ).

check_single_deficiency(mobility_failure) :-
    required_mobile(true),
    \+ robot_mobile(true).

check_single_deficiency(insufficient_arms(Required, Actual)) :-
    required_arms(Required),
    robot_arms(Actual),
    Actual < Required.

check_single_deficiency(insufficient_dof(Required, Actual)) :-
    required_dof(Required),
    robot_dof(Actual),
    Actual < Required.

check_single_deficiency(missing_gripper) :-
    required_gripper(true),
    \+ (robot_arms(A), A >= 1).

check_single_deficiency(missing_rigid_grip) :-
    required_rigid(true),
    \+ (robot_arms(A), A >= 1).

```

A.5. PIPELINE 2 PROLOG PROGRAMS

```
check_single_deficiency(minimum_capability_failure) :-
    task_needs_contact(false),
    \+ task_needs_grasping(true),
    robot_arms(A), A >= 1,
    robot_dof(D), D < 3.
```

A.5.4 Table Setting Cutlery and Plate Reasoner

The Prolog program for selecting cutlery and plates is structured as follows:

```
:- dynamic is_soup_or_broth/1.
:- dynamic is_handheld_food/1.
:- dynamic needs_cutting/1.
:- dynamic is_asian_cuisine/1.
:- dynamic is_long_noodles/1.
:- dynamic has_sauce_or_small_pieces/1.
:- dynamic is_dessert/1.
:- dynamic is_skewer_food/1.
:- dynamic is_tongs_food/1.

% Cutlery Rules
cutlery('Spoon') :-
    is_soup_or_broth(true).
cutlery('Spoon') :-
    is_dessert(true),
    is_handheld_food(false).
cutlery('Spoon') :-
    has_sauce_or_small_pieces(true),
    is_long_noodles(false),
    is_handheld_food(false),
    needs_cutting(false),
    is_asian_cuisine(false).
cutlery('Spoon') :-
    is_long_noodles(true),
    is_asian_cuisine(false).

cutlery('Fork') :-
    needs_cutting(true).
cutlery('Fork') :-
    is_long_noodles(true).
cutlery('Fork') :-
    is_dessert(true),
    is_handheld_food(false).
cutlery('Fork') :-
```

```

    is_soup_or_broth(false),
    is_dessert(false),
    is_handheld_food(false),
    is_asian_cuisine(false).

cutlery('Knife') :-
    needs_cutting(true).
cutlery('Knife') :-
    is_long_noodles(true),
    is_asian_cuisine(false).
cutlery('Knife') :-
    is_soup_or_broth(false),
    is_dessert(false),
    is_handheld_food(false),
    is_asian_cuisine(false),
    has_sauce_or_small_pieces(false),
    is_long_noodles(false).

cutlery('Chopsticks') :-
    is_asian_cuisine(true),
    (is_long_noodles(true) ;
     has_sauce_or_small_pieces(true) ;
     is_handheld_food(true)).

cutlery('Hands') :-
    is_handheld_food(true).

cutlery('Fork') :-
    is_handheld_food(true),
    is_dessert(false),
    is_asian_cuisine(false).
cutlery('Knife') :-
    is_handheld_food(true),
    is_dessert(false),
    is_asian_cuisine(false).

cutlery('Fork') :-
    is_soup_or_broth(true),
    is_asian_cuisine(false).

cutlery('Skewer') :-
    is_skewer_food(true).
cutlery('Tongs') :-
    is_tongs_food(true).

```

A.5. PIPELINE 2 PROLOG PROGRAMS

```
% Retrieve all valid cutlery as a list without duplicates.
get_all_cutlery(Result) :-
    setof(C, cutlery(C), Result),
    !.
get_all_cutlery([]).

% Plate Rules
plate('Bowl') :-
    is_soup_or_broth(true), !.
plate('Dessert Plate') :-
    is_dessert(true), !.
plate('Bowl') :-
    is_long_noodles(true), !.
plate('Bowl') :-
    has_sauce_or_small_pieces(true),
    is_handheld_food(false), !.
plate('Dinner Plate') :-
    \+ is_soup_or_broth(true),
    \+ is_dessert(true),
    \+ is_long_noodles(true),
    \+ has_sauce_or_small_pieces(true), !.
plate('Dinner Plate').

evaluate_meal :-
    get_all_cutlery(CutleryList),
    plate(Plate),
    write('CUTLERY='), write(CutleryList), nl,
    write('PLATE='), write(Plate), nl,
    halt.
```

A.5.5 Procedural Knowledge Temporal Reasoner

The Prolog program for recipe step temporal ordering is structured as follows:

```
:- dynamic step_a_phase/1, step_b_phase/1.
:- dynamic step_a_requires_heat/1, step_b_requires_heat/1.
:- dynamic step_a_is_final/1, step_b_is_final/1.
:- dynamic step_a_transforms_state/1, step_b_transforms_state/1.
:- dynamic step_a_depends_on_b/1, step_b_depends_on_a/1.
:- dynamic step_a_belongs_to_recipe/1, step_b_belongs_to_recipe/1.

% Gate check
valid_steps :-
    step_a_belongs_to_recipe(true),
    step_b_belongs_to_recipe(true).
```

```

% Phase ordering hierarchy
phase_order(preheating, 1).
phase_order(prepare, 2).
phase_order(mixing, 3).
phase_order(marinating, 4).
phase_order(heating, 5).
phase_order(cooking, 6).
phase_order(assembly, 7).
phase_order(resting, 8).
phase_order(cooling, 9).
phase_order(finish, 10).
phase_order(serving, 11).

% Rule 1: Phase-based ordering
comes_before :-
    valid_steps,
    step_a_phase(PA), step_b_phase(PB),
    phase_order(PA, OA), phase_order(PB, OB),
    OA < OB.

% Rule 2: Dependency-based ordering
comes_before :-
    valid_steps,
    step_b_depends_on_a(true).

% Rule 4: Final step heuristic
comes_before :-
    valid_steps,
    step_a_is_final(false), step_b_is_final(true).

% Rule 5: Heat progression
comes_before :-
    valid_steps,
    step_a_phase(PA), step_b_phase(PB), PA == PB,
    step_a_requires_heat(false), step_b_requires_heat(true).

% Rule 6: State transformation
comes_before :-
    valid_steps,
    step_a_phase(PA), step_b_phase(PB), PA == PB,
    step_a_transforms_state(false),
    step_b_transforms_state(true).

```