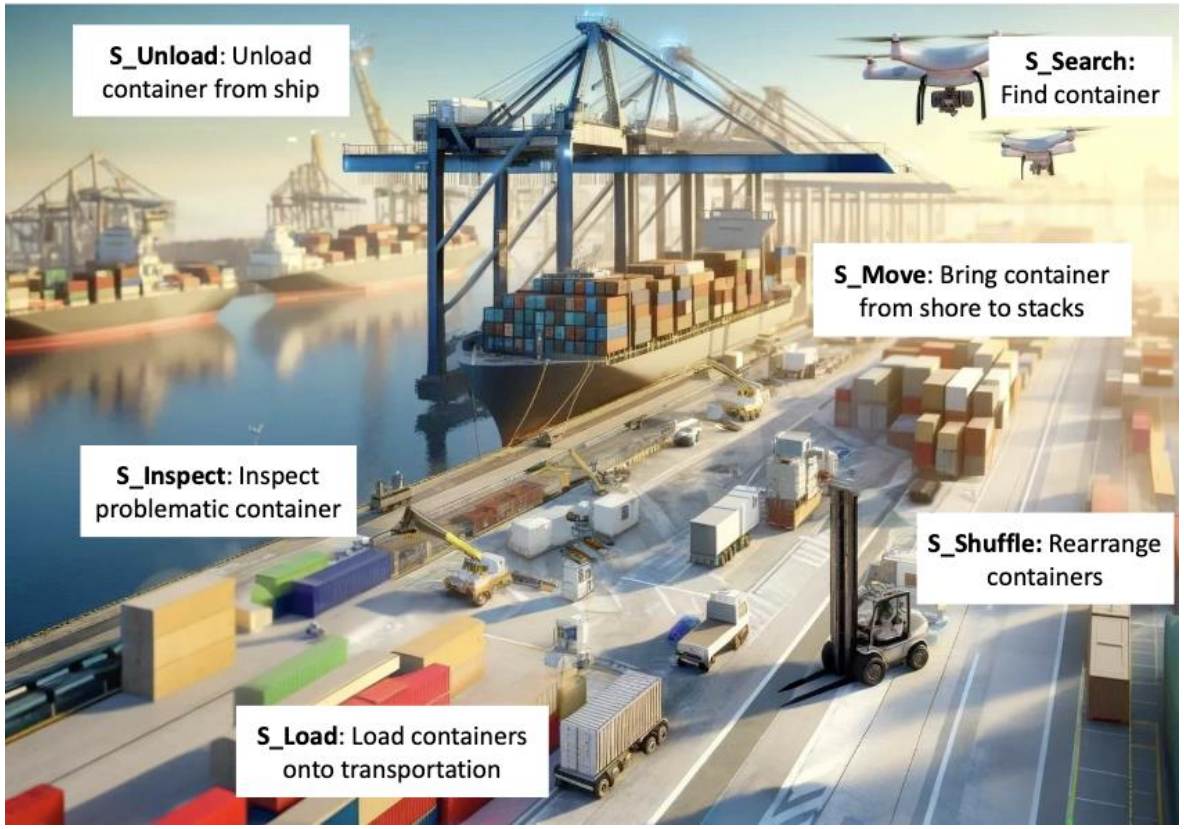
Tampere University

SysML v2 and Gen AI for trustworthy multi-robot systems

Motivation & Context

Mixed Fleet: Multiple autonomous machines from different **types,** and **various manufacturers** cooperate in the same workflow.



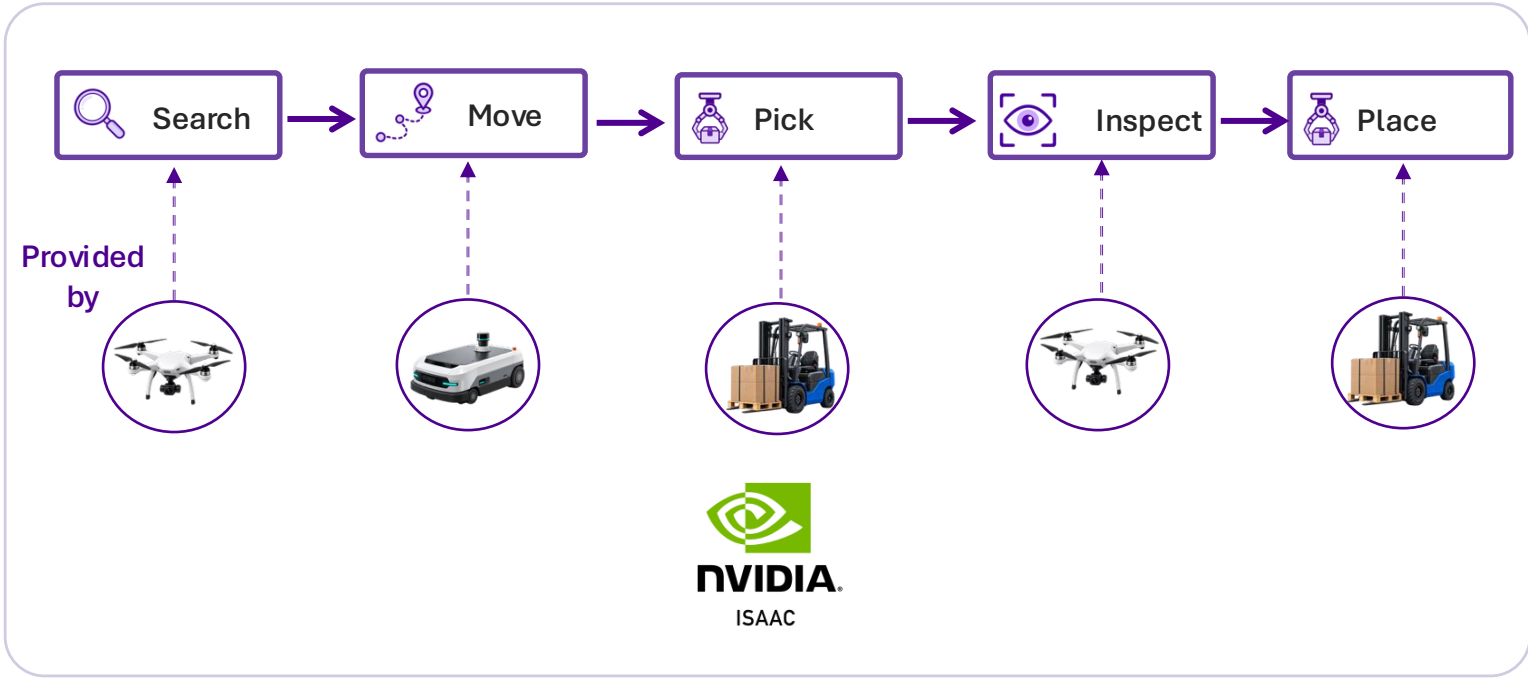
B. Wiesmayr, A. Zoitl, and D. Hästbacka, “Modeling service choreographies and collaborative tasks for autonomous mixed-fleet systems,” in Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems, MODELS Companion’24

Could LLM help ?

From natural language to mission execution

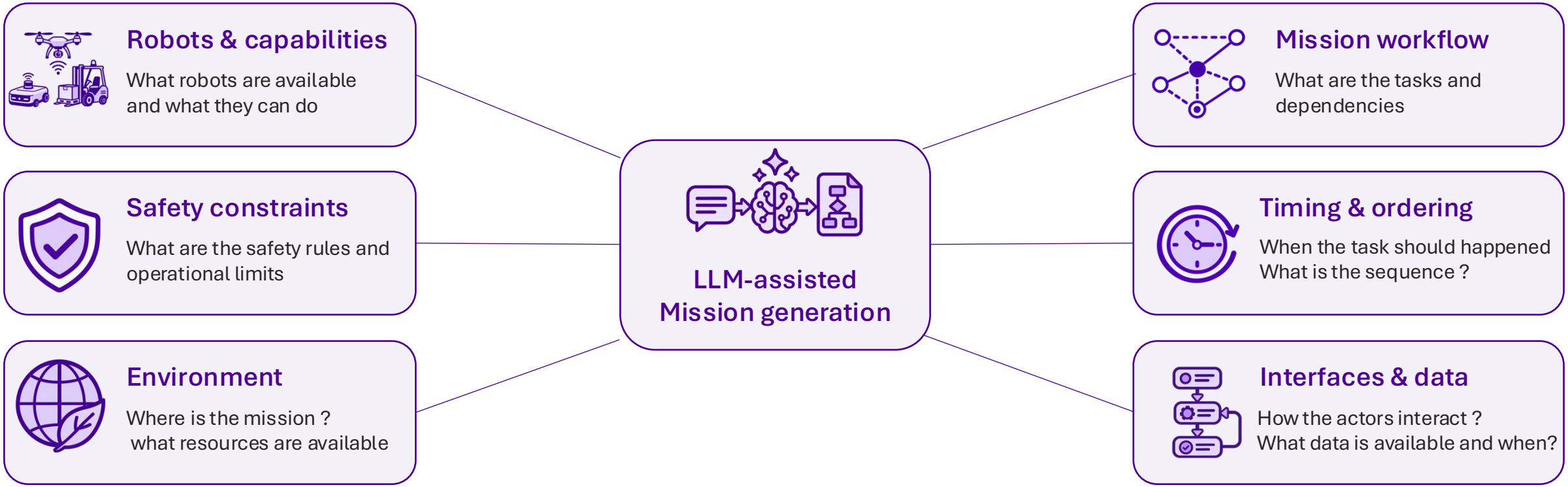
Scan rack A, then move pallet 3 to the loading area. The forklift must not start pickup before the scan is complete.

LLM



But what does the LLM need to know ?

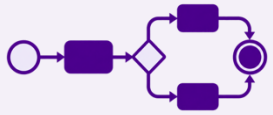
How much context is enough to generate a mission that is correct



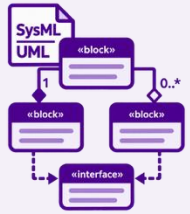
A single source of truth

We need the right context

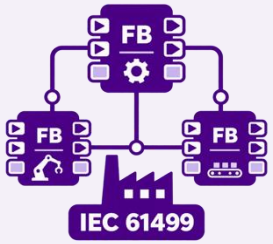
How mission can be defined



BPMN



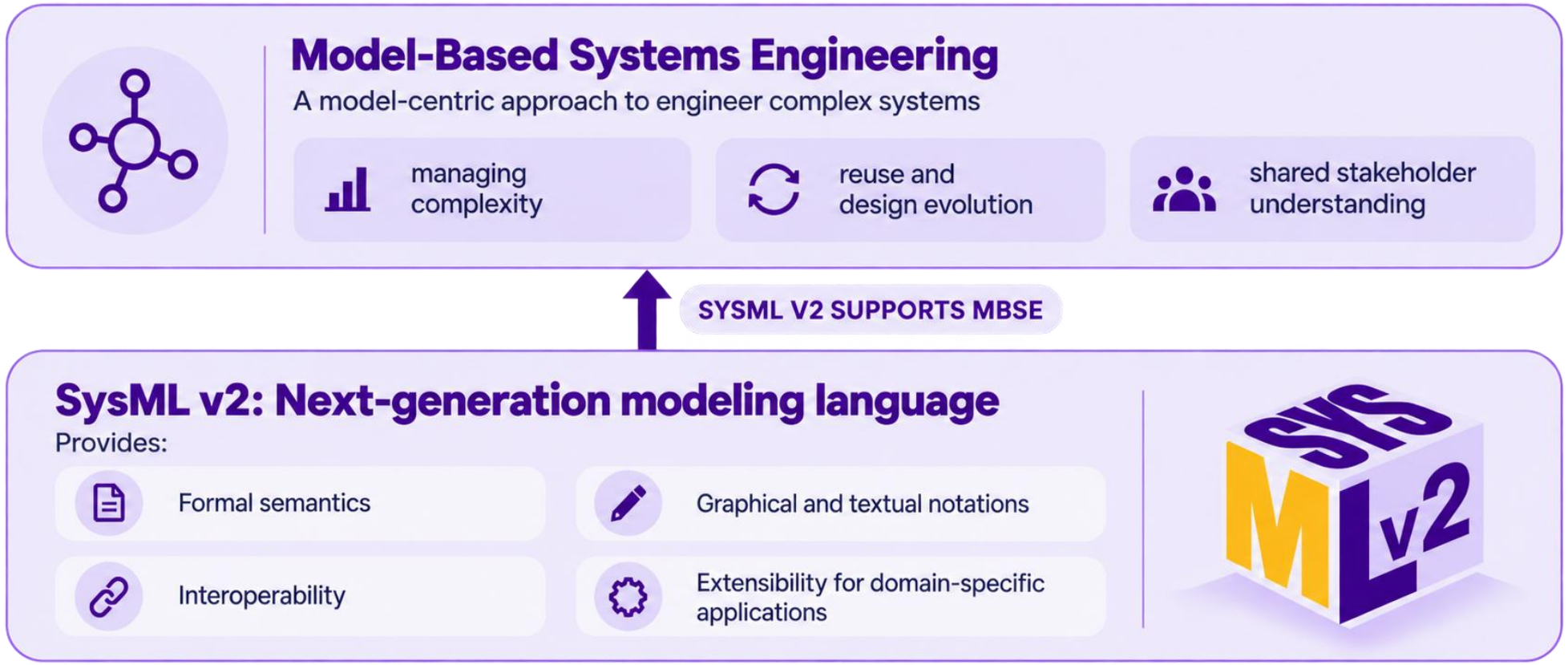
SysML/UML



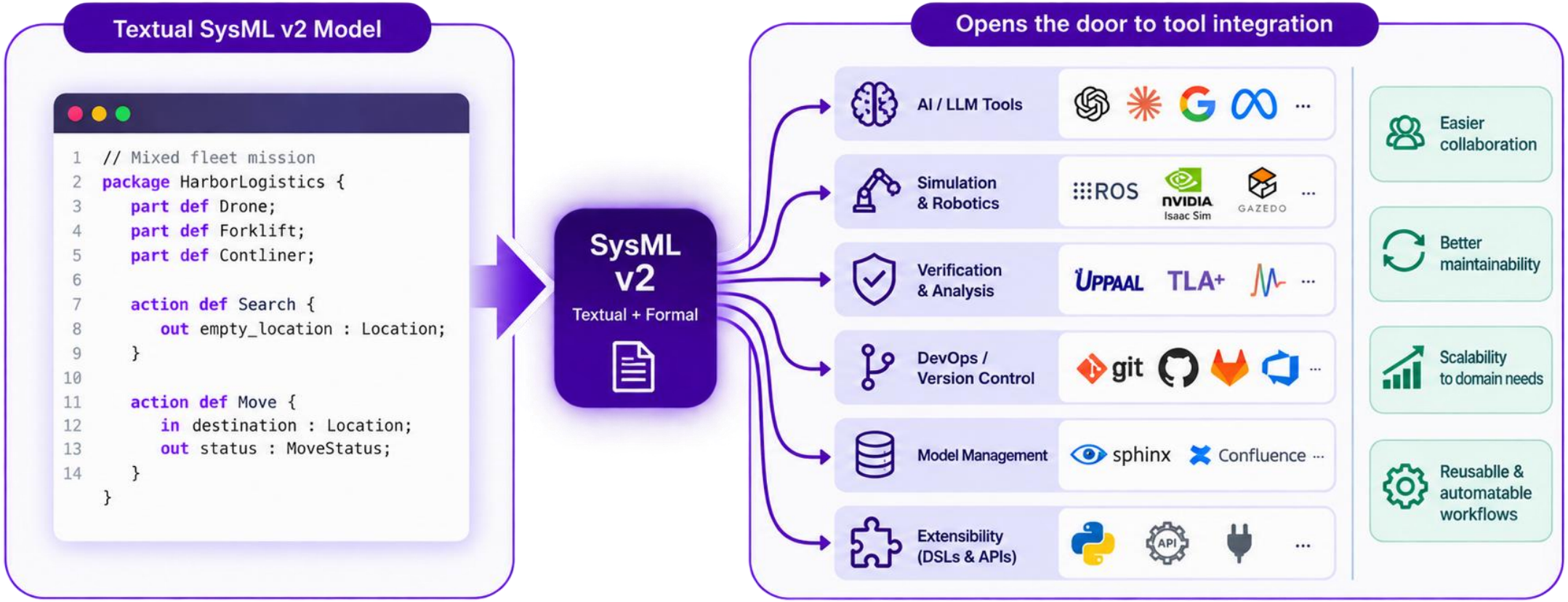
IEC 61499

SysMLv2

SysML v2 what's new ?



Context is king when selecting a tool



What about the language, is it expressive ?

Coming from SysML 1.X background things will be different

Concept	SysML v2 Construct	Concrete Example in the Model
Requirement (abstract capability)	`requirement def`	RD_Move — general capability to move an item
Requirement (contextual instance)	`requirement usage`	R_Move_Container — bound to a specific AGV + container
Actor / vehicle	`part def / part`	Agv01 : Agv — liftingCapacity = 1000 kg
Service	`action def`	S_Move — the container-move operation
Precondition / assumption	`constraint (assume / require)`	destinationIsDifferentFromSource
Postcondition / proof	`verification case + verify`	VehicleMovementTest verifies R_Move_Container
Reuse hierarchy	`specializes`	Crane, AGV, UAV specialize a common Vehicle
Execution link	`perform`	Agv01 performs moveContainer : S_Move

Definitions vs Usages

Type/instance pattern

Definition

Defines what a thing is (template)

- Abstract type
- Reusable definitions

Usage

Defines a specific occurrence in a system context

- concrete
- Type enforcement

Block Explosion

Why SysML v1.X block counts multiply, SysML v2 fixed it !

SysML v1 — A Block per Variant

Every variation becomes a new Block

near-duplicate structure + more diagrams

«block»
Vehicle

«block»
VehicleStd
standard config

«block»
VehicleEV
EV config

«block»
VehicleTeleop
teleoperated config

«block»
VehicleAuto
autonomous config

SysML v2 — One Definition, Many Usages

same reusable definition, configured per usage

part def
Vehicle

```
part def Engine;
part def Wheels;
action def Start;
action def Stop;
```

part **myCar01** : Vehicle | standard config

part **myCar02** : Vehicle | EV config

part **myCar03** : Vehicle | teleoperated config

part **myCar04** : Vehicle | autonomous config

Tampere University

Handling variation

We just re-configure no new block needed

SysML v2 (Textual Syntax)

```
1  part def Forklift {
2      attribute maxSpeed : Speed;
3  }
4
5  part indoorForklift : Forklift {
6      :>> maxSpeed = 30[km/h];
7  }
```

redefines in context

Result in Context

The re-usable definition remains the same, the change is introduced at the usage level

Also in requirements

Mission specific requirement

Requirement definition

reusable template

```
requirement def RD_Move specializes EnvironementRequirementDef {
    doc /* Move item from location to destination */

    assume itemIdIsValid {
        :>> itemId = itemId;
    }

    require destinationIsDifferentFromSource {
        :>> source = fromLocation;
        :>> destination = toLocation;
    }
}
```

- Defines a generic move requirement
- Captures reusable assumptions
- Captures reusable constraints

specialize + bind in context

Requirement usage

context-specific instance

```
requirement <'S.2'> R_Move_Container : RD_Move {
    doc /* Move container from shore to stack by the Agv */

    subject agv : Agv;
    stakeholder operator : Person;
    actor environment : Environment;
    item container : Container;

    attribute :>> itemId = container.id;
    attribute :>> fromLocation = environment.shoreLocation;
    attribute :>> toLocation = environment.storageArea;
}
```

- subject: agv
- stakeholder: operator
- actor: environment
- item: container
- binds abstract attributes to context values

Closing the loop

Context aware missions ?

SysML v2 (Action)

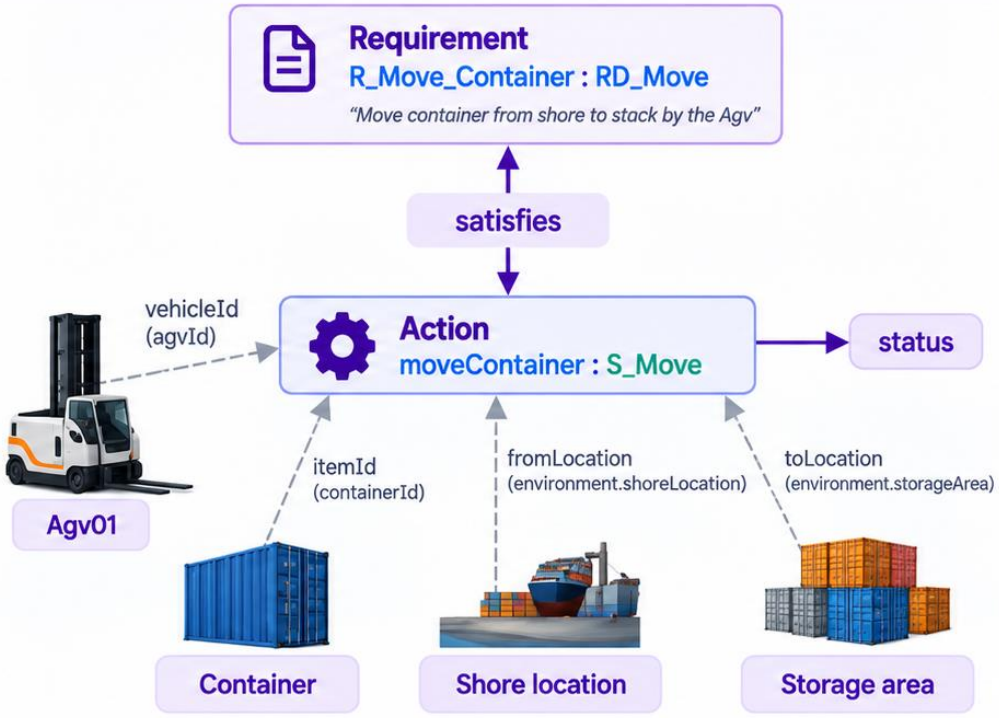
Connect actions to requirements using satisfy ... by

```
1   action moveContainer: S_Move{
2
3       in :>> vehicleId = agvId;
4       in :>> itemId = containerId;
5       in :>> fromLocation =
6               environment.shoreLocation;
7       in :>> toLocation =
8               environment.storageArea;
9       out status;
10      satisfy R_Move_Container by Agv01;
11  }
```

Links this action to a requirement and the responsible actor

Result in Context

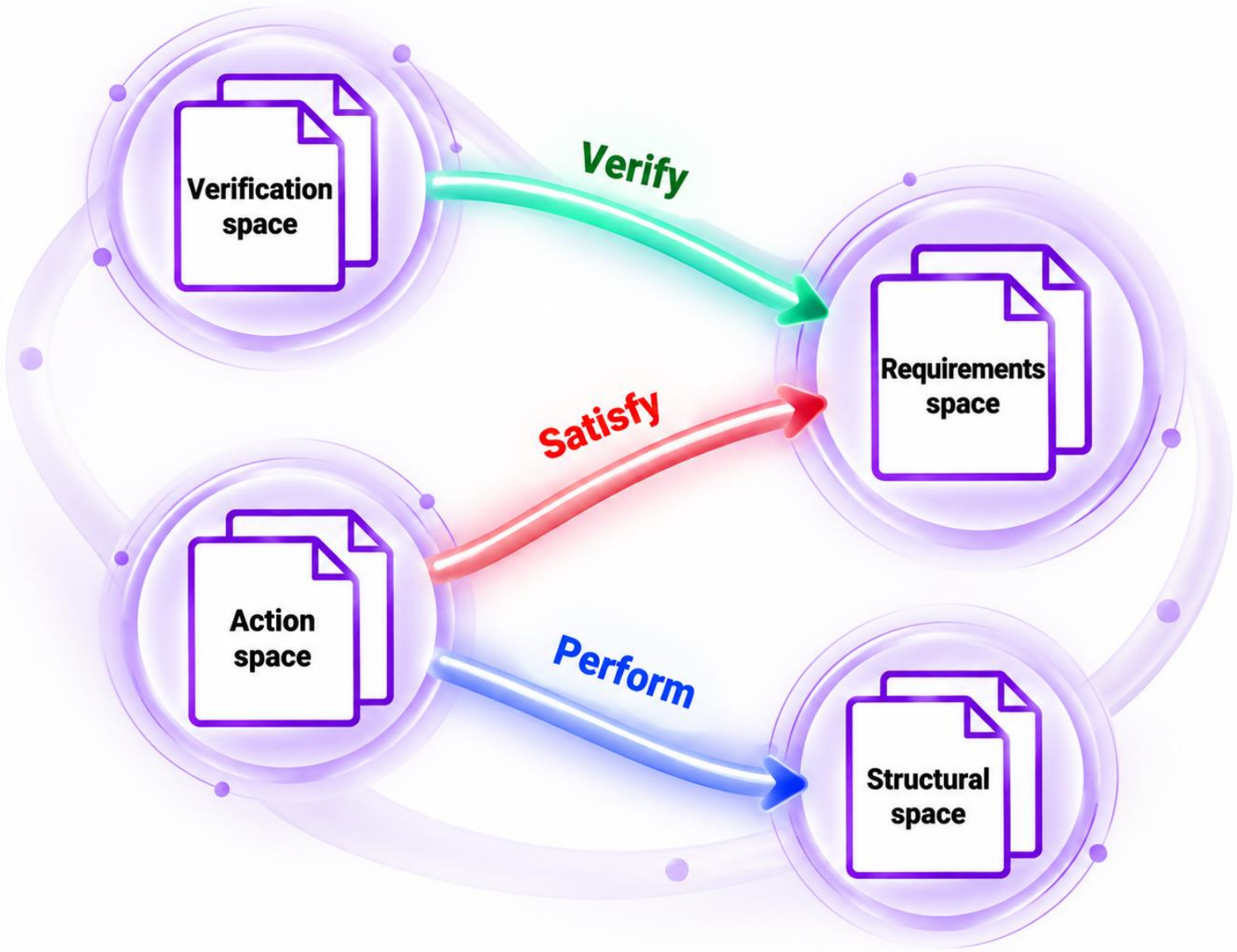
The action satisfies the requirement, enabling traceability



Traceability by Design

Actions explicitly satisfy requirements, linking behavior to intent, actors and context

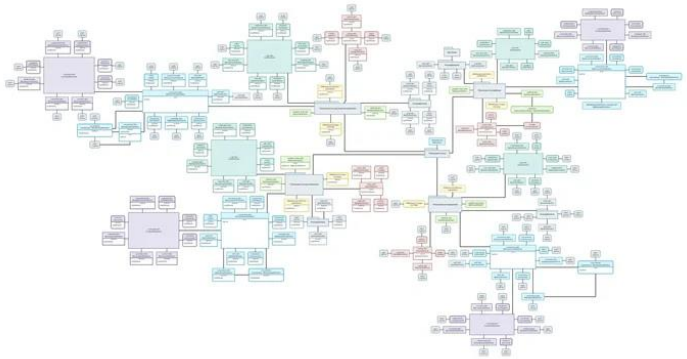
The pattern

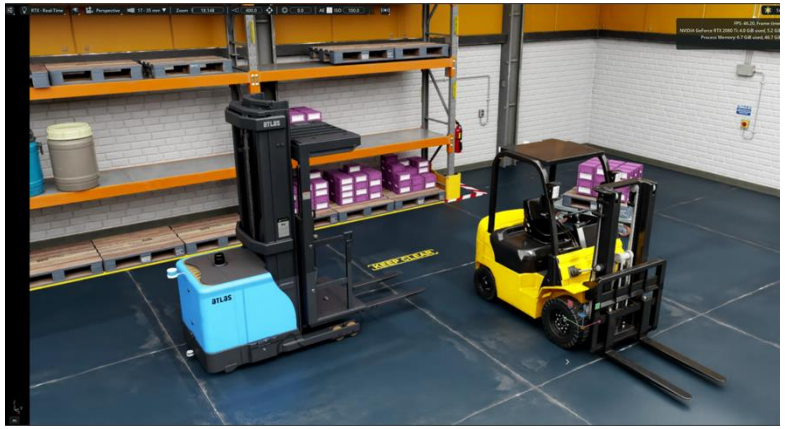

Verification space
Verify
Requirements space
Satisfy
Action space
Perform
Structural space

From Models to Execution

Making models operational

SysML v2 Mission Model → Transformation Paser / Api → Intermediate representation → Mission Orchestrator → MQTTT → Simulation





State-Space Explosion

Why manual reasoning does not scale

Global mission state = $S_1 \times S_2 \times \ldots \times S_n \times S_e$

As the fleet grows, the joint state space grows combinatorially.

2 robots, 5 states each → **25 global states**

5 robots, 5 states each → **3,125 global states**

10 robots, 5 states each → **~ 9.8 million global states**

+ environment states, timing, shared resources, and failure modes

The number of possible mission situations quickly exceeds what a domain expert can inspect manually.

The Core Problem

Why manual reasoning does not scale

How the plan is created

SySML models

Natural Language

Plan Correctness and Safety gap

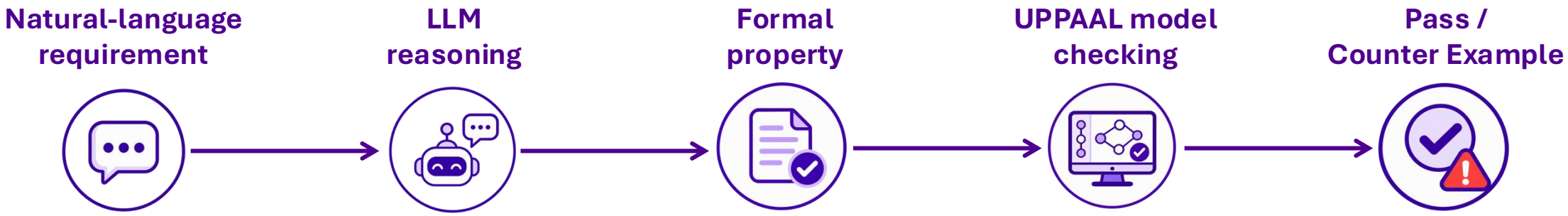
What must be handled

- Mission dependencies and synchronization
- Environment and safety constraints
- Many possible execution paths

We need methods that let domain experts specify missions at a high level while engineering tools checks plan correctness

Result 3 – Verify Mission Safety

From NL to formal checks



Example

The forklift places pallets after drone scanning is completed. Also the forklift and drone must never move in the same time.

Safety property Example

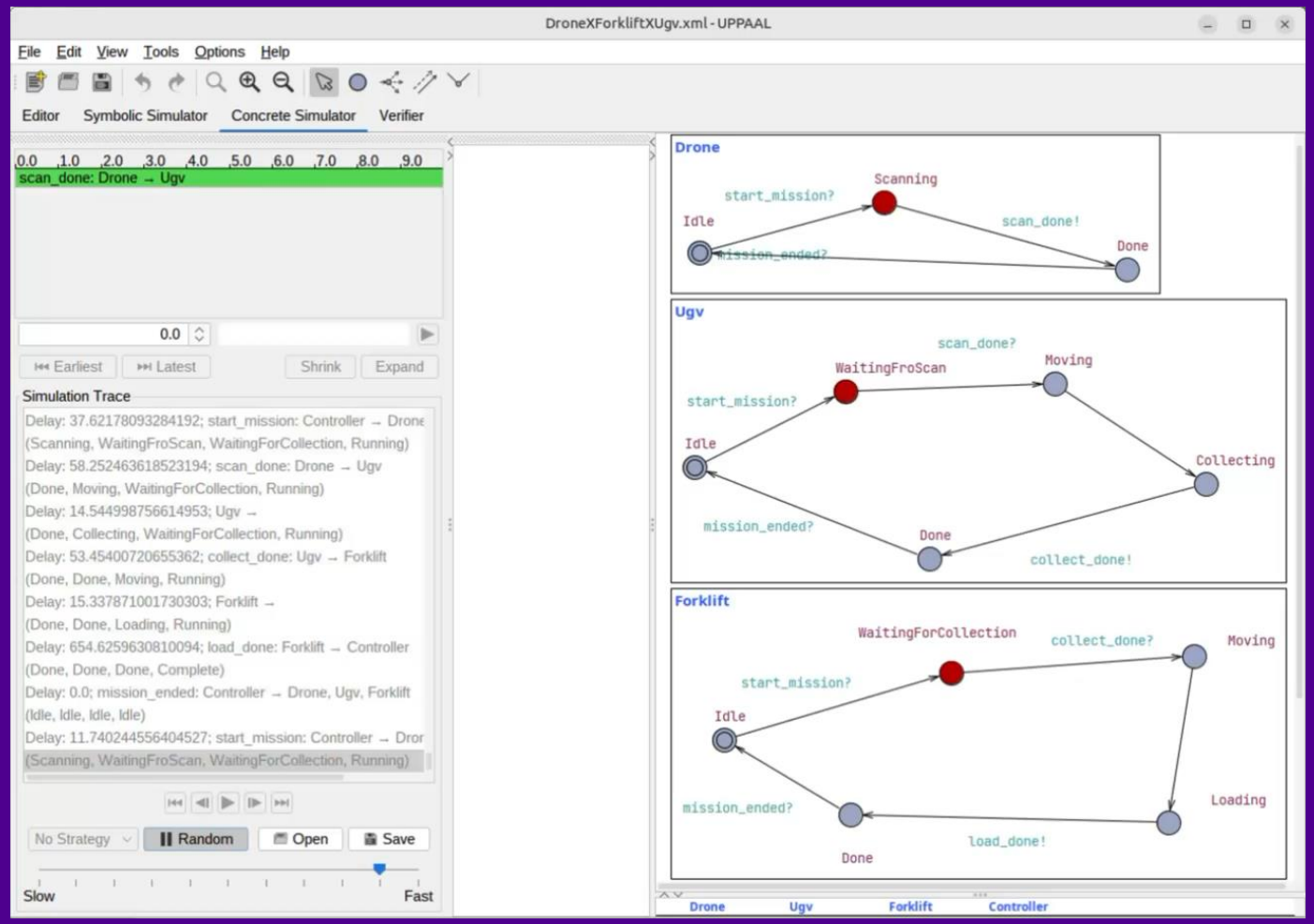
```
A[] (Forklift.Placing imply Drone.scanning_done)

A[] not (Forklift.Moving and Drone.Moving)
```

Observed LLM failure modes

- Missing constraints
- Hallucinated constraints
- Incorrect temporal relationships
- Overgeneralization

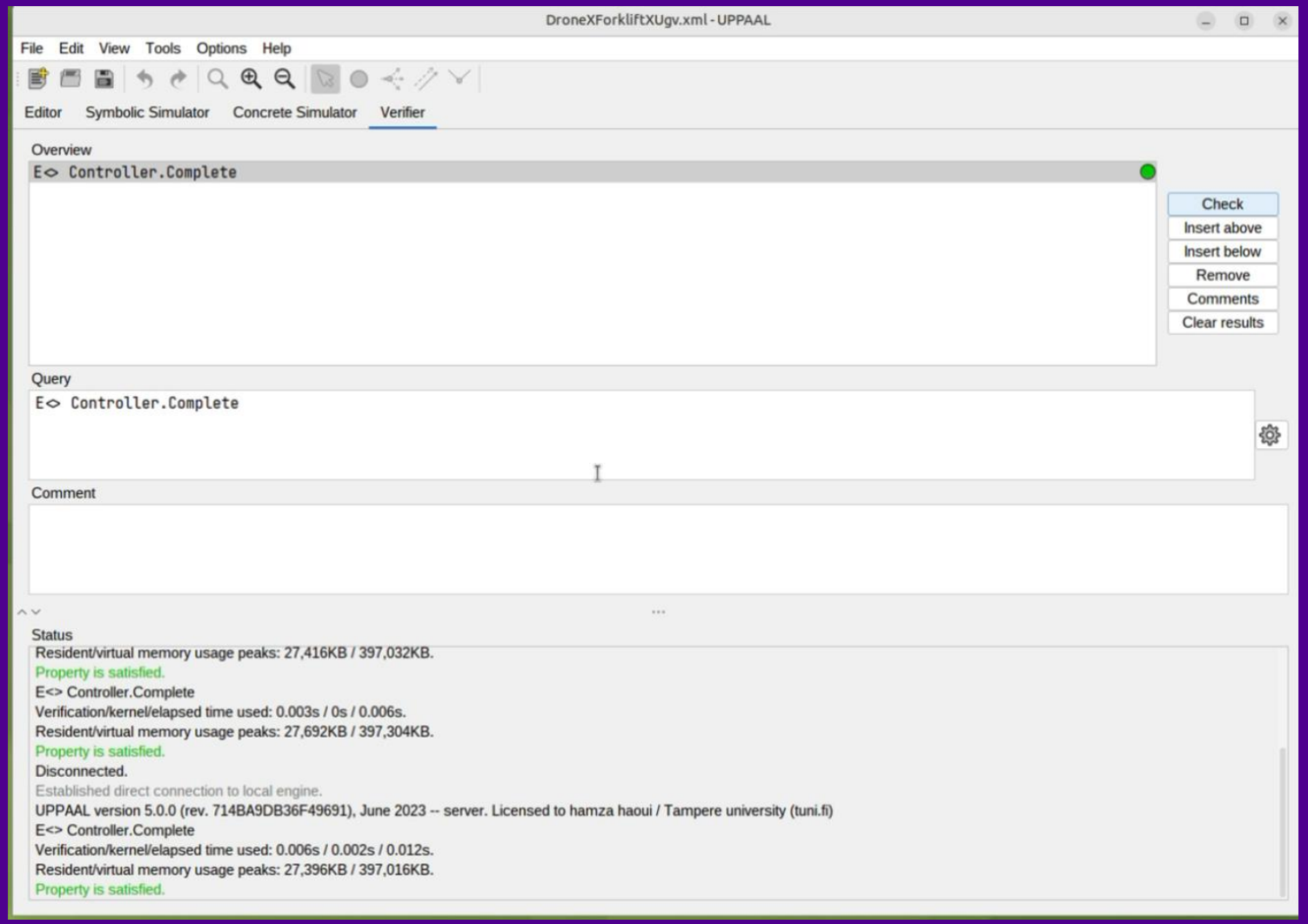
UPPAAL – model checker

- Based on the literature Model Checking is the most used formal verification tool. State-transition systems are the most numerous for specifying robotics systems.



UPPAAL – model checker

- Based on the literature Model Checking is the most used formal verification tool. State-transition systems are the most numerous for specifying robotics systems.



Are LLMs Effective in Fromal Verification Tasks ?

We created NL2TCTL data set that contains 40 robotics mission scenario and +1900 generated formal verification queries

Benchmarked 12 state of the art LLMs on both tasks :

1. **Extracting constraints**
2. **Formalizing the constraints** into UPPAAL queries (Both syntax and semantic levels)

```
==================== MODEL RANKING (F1) ====================

1. openai::gpt-5-mini                          F1 = 0.969
2. anthropic::claude-sonnet-4-5-20250929       F1 = 0.953
3. anthropic::claude-haiku-4-5-20251001        F1 = 0.944
4. openai::gpt-5-nano                          F1 = 0.930
5. deepseek::deepseek-reasoner                 F1 = 0.907
6. openai::gpt-5.1                             F1 = 0.907
7. google::gemini-2.5-pro                      F1 = 0.898
8. openai::gpt-4o                              F1 = 0.890
9. xai::grok-code-fast-1                       F1 = 0.805
10. xai::grok-3-mini                            F1 = 0.803
11. google::gemini-2.0-flash                    F1 = 0.755

============================================================
```

Effectiveness in extracting atomic constraints from scenario –prompt 2

```
🏆 MODEL RANKING BY VALIDITY RATE
```

Rank	Provider	Model	Validity	Valid/Total	Scenarios
1	google	gemini-2.5-pro	100.00%	160/160	40
2	openai	gpt-5-mini	98.80%	165/167	40
3	anthropic	claude-sonnet-4-5-20250929	98.35%	179/182	40
4	openai	gpt-5.1	96.34%	158/164	40
5	xai	grok-3-mini	96.27%	155/161	40
6	anthropic	claude-haiku-4-5-20251001	95.93%	165/172	40
7	openai	gpt-4o	95.65%	154/161	40
8	deepseek	deepseek-reasoner	94.48%	154/163	40
9	google	gemini-2.0-flash	94.01%	157/167	40
10	xai	grok-code-fast-1	93.53%	159/170	40
11	openai	gpt-5-nano	93.02%	160/172	40

```
📊 SUMMARY STATISTICS
  Total Models: 11
  🥇 Best Model: google/gemini-2.5-pro (100.00%)
  Average Model Validity: 96.04%
  Validity Range: 93.02% - 100.00%
  Total Properties Evaluated: 1839
```

Effectiveness in Formalizing Syntax level

```
============== PROPERTY GENERATION MODEL RANKING (PRECISION) ==============

1. anthropic::claude-sonnet-4-5-20250929    Precision = 0.739
2. openai::gpt-5.1                          Precision = 0.733
3. google::gemini-2.5-pro                   Precision = 0.691
4. openai::gpt-4o                           Precision = 0.627
5. google::gemini-2.0-flash                 Precision = 0.558
6. openai::gpt-5-mini                       Precision = 0.551
7. deepseek::deepseek-reasoner              Precision = 0.549
8. xai::grok-3-mini                         Precision = 0.438
9. anthropic::claude-haiku-4-5-20251001     Precision = 0.404
10. xai::grok-code-fast-1                    Precision = 0.254
11. openai::gpt-5-nano                       Precision = 0.206

========================================================================
```

Effectiveness in Formalizing Semantics level

Objective

Core Mission

Transform natural language constraints into formally verified UPPAAL temporal logic queries through intelligent pattern recognition and deterministic retrieval.

VALUE PROPOSITION

Empowers non-technical domain experts to express complex system constraints in plain language while ensuring formal verification query correctness

Transformation Example

Example

IN **Natural Language Input**

Input

"Drone should not be flying when Forklift is moving "

OUT **UPPAAL Query Output**

Output

A[] (Drone.Flying imply NOT Forklift.Moving)

LLM Classification

STEP 1

Pattern Identification

Classifies input into 8 temporal patterns (e.g., **conditional_response**).

Classification Output

Output

Structured JSON object with pattern type, keywords, and temporal cues:

```
{
  "pattern": "conditional_response",
  "keywords": ["battery", "base"],
  "trigger": "battery < 10",
  "response": "drone.location == base"
}
```

Key Insight

This step transforms unstructured language into structured data, enabling deterministic retrieval.

Neo4j Knowledge Graph

STEP 2

Pattern Template Retrieval

Fetches template for conditional_response : A[] (condition imply action)

Operator Semantics

Retrieves **A[]** (always) and **imply** operators with usage rules.

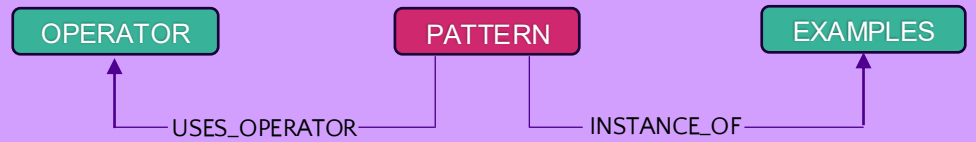
Few-Shot Examples

Gets similar examples like **"Robot stops when obstacle detected"**.

GraphRAG Advantage

Deterministic retrieval ensures consistency. Pattern templates guarantee correct structure every time.

Graph Schema Relationship



LLM Generation

STEP 3 3

1 **Template Application**

GPT applies **"A[] (condition imply action)"** to the constraint.

2 **Operator Guidelines**

Uses operator semantics to ensure correct syntax for **A[]** and **imply**.

3 **Few-Shot Learning**

Leverages similar examples to improve generation quality and consistency.

Generated Query:

```
A[] (battery < 10 imply drone.location == base)
```

ANTLR Validation

STEP 4

Grammar Parsing

ANTLR parses the query against the formal UPPAAL grammar to verify structure.

Syntactic Verification

Checks for proper nesting, balanced parentheses, and operator precedence.

Error Detection

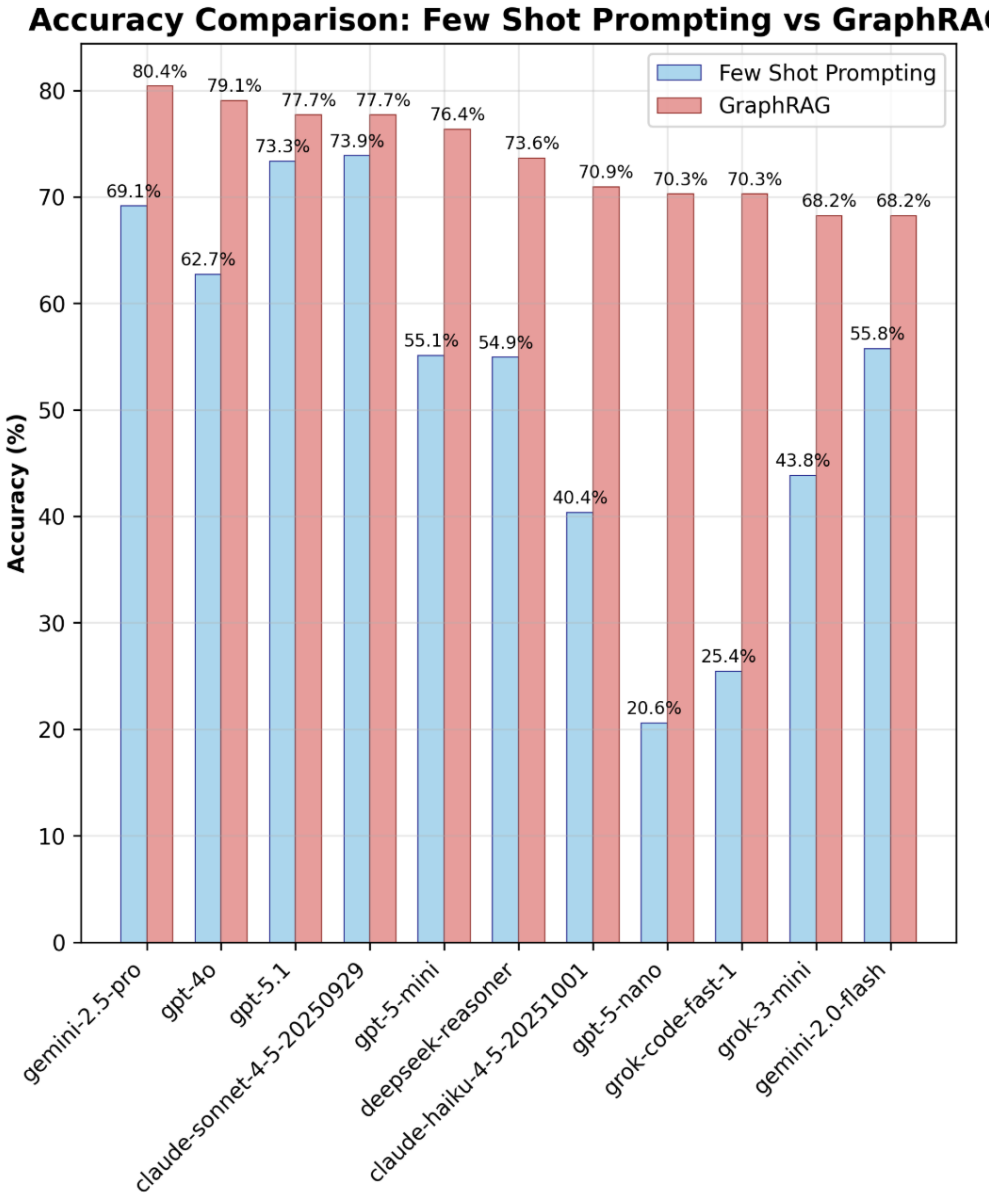
Identifies syntax errors, typos, or grammar violations with detailed messages.

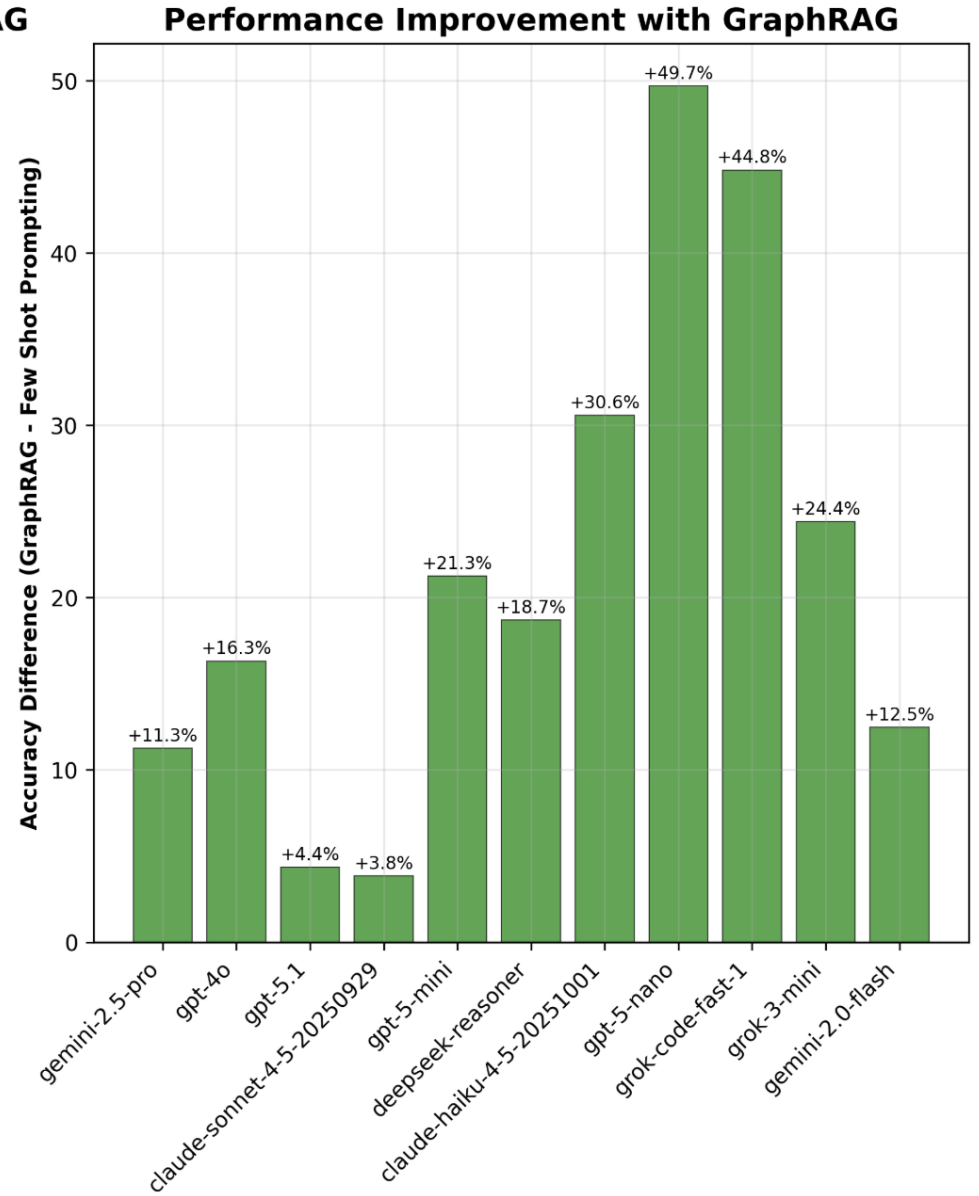
Validation Output:

```
{
"antlr_ok": true,
"errors": []
}
```

Tampere University

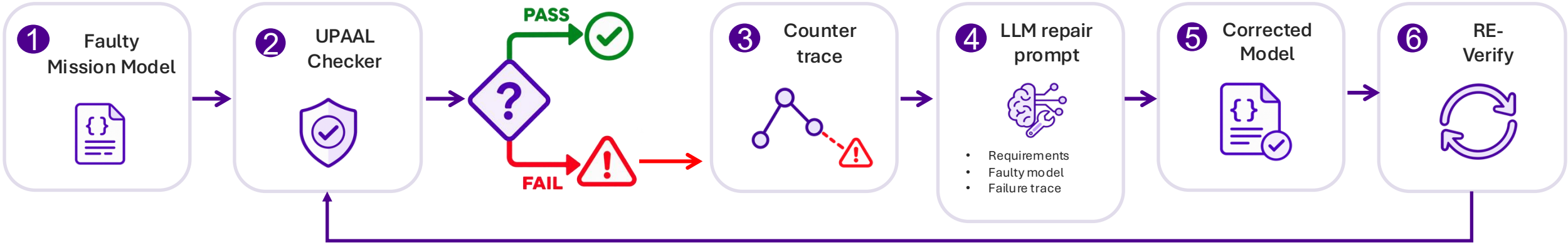
Results





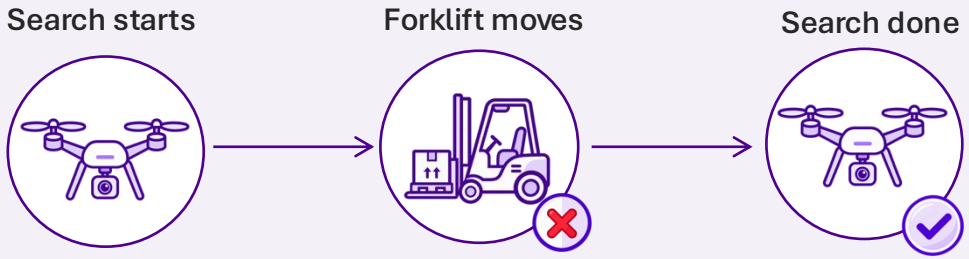
Result 4 – From Verification to Repair

Using counterexample traces to guide LLM-based correction



Key idea

”Forklift must not move before drone search completes ”



The counterexample reveals the violating execution order

Formal verification not only a checker, but a feedback source for automated mission repair

Result 4 – LLMs and counter-examples

Concrete failure evidence can reduce repair ambiguity ?

Without counterexample trace

Requirements +faulty model → LLM → Repair attempt

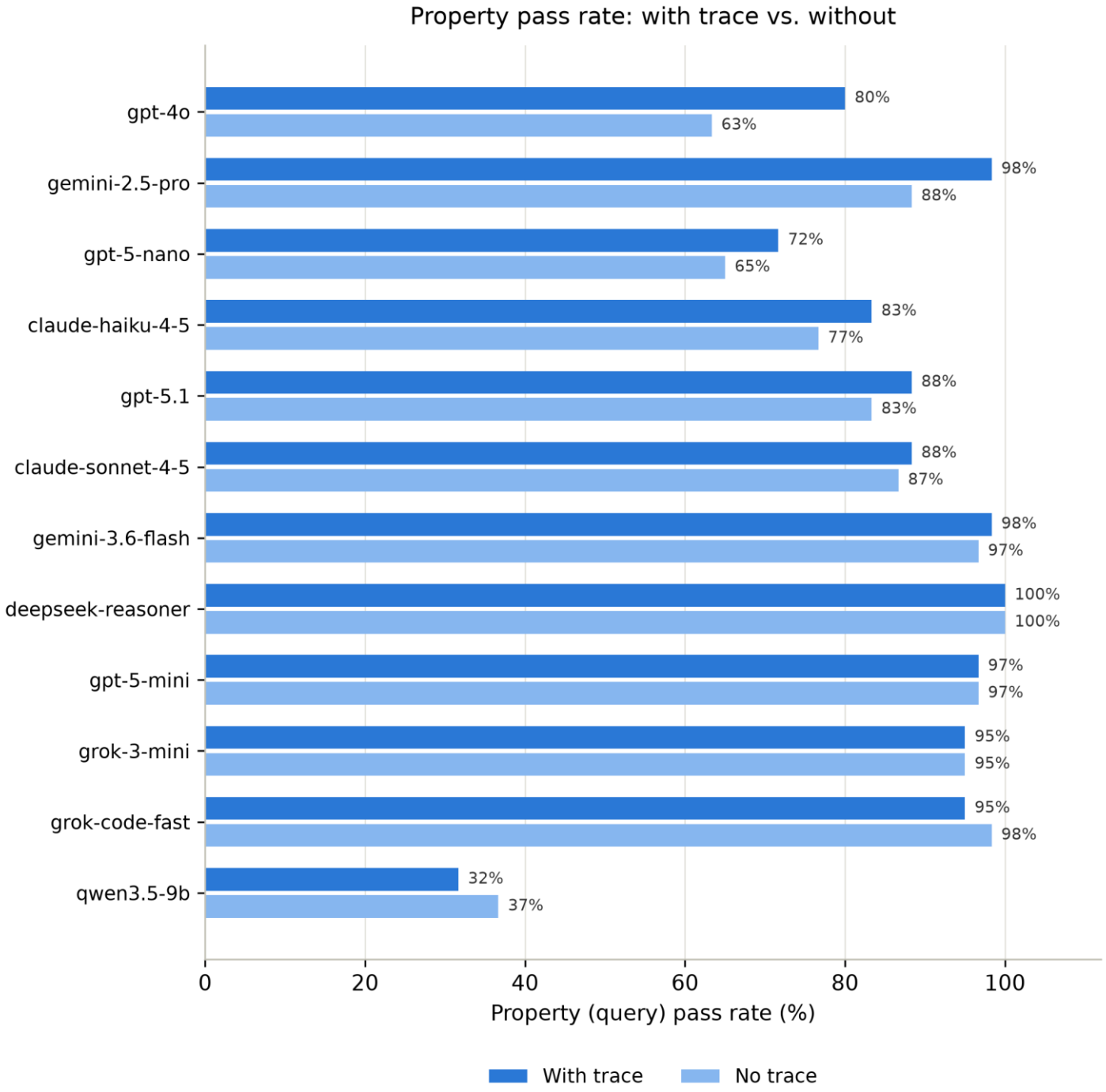
LLM need to guess the cause of failure

With counterexample trace

Requirements + faulty model + traces → LLM → Repair attempt

LLM receives the violating execution path

And we observe



What the data says

+3.3 average gain in pass rate points

7/12 models improved with trace

3/12 models unchanged

2/12 models degraded

Largest improvement: GPT-4o (+16.7 pts)

Counterexample traces usually help, but the effect depends on whether the LLM can interpret the formal execution trace correctly.

More information

hamza.haoui@tuni.fi
elmeri.pohjois-koivisto@tuni.fi
david.hastbacka@tuni.fi
kari.systa@tuni.fi