

Trustworthy AI for Disaster Resourcing Using Large Language Models and Formal Verification

An applied research project submitted to Auckland University of Technology
in partial fulfilment of the requirements for the degree of
Master of Computer and Information Sciences

Caleb Elson

School of Computer Science and Software Engineering
Auckland University of Technology
Auckland, New Zealand
`sbh2324@autuni.ac.nz`

November 5, 2025

Abstract

Formal verification provides a rigorous foundation for evaluating correctness in systems operating under uncertainty, yet it remains largely inaccessible to non-specialists due to its mathematical and structural complexity. This dissertation investigates how large language models (LLMs) can be integrated with probabilistic model checking to make formal verification more usable in disaster resourcing contexts. A modular PRISM-based pipeline was developed that converts natural language scenario descriptions into structured JSON, composes verifiable Markov Decision Processes (MDPs), and computes optimal strategies using probabilistic model checking and Dijkstra’s algorithm. The pipeline emphasises transparency and reproducibility, recording every artefact from model generation to verification outcomes. It combines the strengths of generative AI and formal verification: LLMs automate model creation and explanation, while PRISM ensures correctness through exhaustive probabilistic analysis. Results show that the pipeline consistently produces verifiable outputs, substantially reducing the technical barrier to formal verification. Although LLMs alone remain unreliable when reasoning about probabilistic outcomes, their integration within a transparent and verifiable workflow demonstrates a practical path toward human-centred formal verification.

Contents

1	Introduction	8
1.1	The PRISM Pipeline	8
1.2	Pipeline Overview	9
2	Related Work	10
3	Parser	12
3.1	Scenario Parsing	12
3.2	Structured Representation with JSON Artefact	12
4	Composer	13
4.1	Model Generation	13
4.2	The Resourcing Problem	13
4.2.1	Algebraic Specification of Resource Distribution	14
4.2.2	Case Study Scenario	14
4.2.3	Resource Allocation and Constraints	15
4.2.4	Teams and Movement Operations	16
4.2.5	Example Strategy and Risk Analysis	17
4.3	Reasoning About Team Safety	18
4.3.1	MDP of the Resource Distribution Model	19
5	Verification	19
5.1	Verification and Strategy Extraction	19
5.1.1	Discrete-Time Markov Chains (DTMCs)	20
5.1.2	Model Verification in PRISM	21
6	Navigation	22
7	Results and Discussion	22
7.1	Overview	22
7.2	Trying to Solve with AI Directly	24
7.2.1	Overview	24
7.2.2	Grok-4	24
7.2.3	Claude 4.1 Opus	25
7.2.4	Gemini 2.5 Pro	25
7.2.5	ChatGPT 5	27
7.2.6	ChatGPT 5 Mini	28
7.2.7	Claude Sonnet 4.5 Thinking	30
7.2.8	Claude Sonnet 4.5	30
7.2.9	Perplexity Sonar	31
7.2.10	DeepSeek V3.2 (DeepThink Enabled)	32
7.3	PRISM Pipeline Results	33
7.3.1	Introduction	33
7.3.2	The PRISM Path	35
7.3.3	Pipeline performance and reproducibility	36

7.3.4 Discussion	37
8 Conclusions and Future Work	37
Appendices	41
A Code	41
B Prompt	41

List of Figures

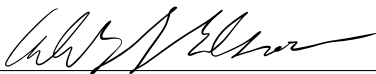
1	Proposed PRISM Pipeline	8
2	Auckland North Shore overlaid with graph G . Maps courtesy of ArcGIS [1].	15
3	Truncated DTMC graph induced from the scenario in Figure 2, showing the optimal strategy obtained from verification of $P_{\max}=?[F(K')]$	20

List of Tables

1	Comparison of models tested using the Mensa Norway and a cus- tom question set, performed 2025-10-20.	11
2	A correct strategy for K'	17
3	Probability of success outputs from various models across three runs.	23
4	Run metrics and strategy summary for Grok 4.	24
5	Run metrics and strategy summary for Claude 4.1 Opus.	25
6	Run metrics and strategy summary for Gemini 2.5 Pro.	26
7	Run metrics and strategy summary for ChatGPT 5.	27
8	Run metrics and strategy summary for ChatGPT 5 Mini.	28
9	Run metrics and strategy summary for Claude Sonnet 4.5 Thinking.	30
10	Run metrics and strategy summary for Claude Sonnet 4.5.	30
11	Run metrics and strategy summary for Perplexity Sonar.	31
12	Run metrics and strategy summary for DeepSeek V3.2 (Deep- Think Enabled).	33
13	PRISM verification metrics for the three full pipeline runs. Ac- tions are the given optimal actions at the end of the pipeline. All runs produced identical state and transition counts, demonstrat- ing structural reproducibility.	33
14	Step-by-step movements and resource allocations along the opti- mal path identified by PRISM.	35
15	End-to-end performance of the PRISM pipeline by stage.	36

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Signed: _____

Name: Caleb Elson

Date: November 5, 2025

Acknowledgements

First and foremost, thank you to my supervisor, Kenneth Johnson, for his guidance, feedback, and patience throughout this research. His knowledge and insight into formal verification and disaster management systems, along with his generosity of time and support, were invaluable in shaping and completing this project.

I would also like to thank the staff and fellow students of the Next Generation Systems group for their support, encouragement, and assistance.

Additionally, I wish to thank the AUT Research Centre for Computer and Information Sciences (RCCIS), which generously supported this research by funding the extensive use of OpenAI tokens during experimentation.

Finally, I am deeply grateful to my friends and family for their understanding and constant motivation throughout my studies. I will try to reply to your messages more promptly in the future.

1 Introduction

Formal verification using probabilistic model checking [2] is a powerful method for decision making under uncertainty. In the context of disaster resourcing, it allows planners to evaluate the probability of success for resource allocation strategies before they are executed, enabling transparent and trustworthy decisions. However, despite its advantages, formal verification remains inaccessible to most domain experts. Constructing models for tools such as PRISM requires substantial mathematical and computational expertise, as well as time-consuming manual specification. These challenges limit the adoption of formal methods in operational settings, where decision making often occurs under significant time pressure.

Large language models (LLMs) offer a promising avenue to improve the usability of formal verification by enabling natural language interaction with complex systems. Rather than requiring a disaster manager to encode a scenario formally, an LLM can interpret a human-readable description and generate a corresponding probabilistic model automatically. This project investigates whether LLMs can make formal verification more accessible without sacrificing rigour. It presents a PRISM-based pipeline that combines the correctness guarantees of model checking with the usability of natural language interfaces. The system is evaluated using a representative disaster resourcing scenario set in Auckland’s North Shore region, illustrated in Figure 2, which serves as the case study throughout this dissertation.

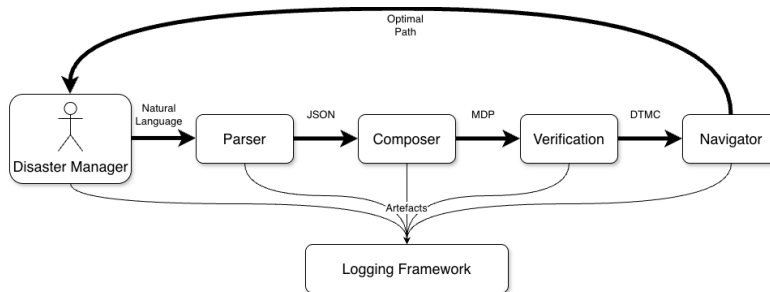


Figure 1: Proposed PRISM Pipeline

1.1 The PRISM Pipeline

In a real-world disaster scenario, being verifiably correct can be the difference between life and death. Current AI systems cannot reliably guarantee correctness, but they can dramatically improve the user experience of model creation. The proposed pipeline integrates LLMs into the formal verification process to combine the strengths of both approaches: PRISM provides the rigorous probabilistic verification, while the LLM automates the tedious and knowledge-intensive steps of model construction. This architecture allows a disaster manager to

describe a situation in natural language and automatically receive a verified optimal strategy for resource distribution.

1.2 Pipeline Overview

The pipeline automates the translation of natural language disaster scenarios into Markov Decision Process (MDP) models for probabilistic verification using the PRISM model checker. It supports trustworthy decision making in disaster management, where uncertainty, risk, and limited resources must be explicitly modelled. As shown in Figure 1, the pipeline follows a modular, sequential design composed of five main stages:

1. **Scenario Input** — the disaster manager provides a natural language description of the scenario, defining available resources, locations, objectives, and the assessed safety of routes connecting those locations.
2. **Scenario Parsing** — an LLM interprets the natural language input, extracting structured information and returning it in a machine-readable JSON format.
3. **Model Composition** — another LLM converts the structured JSON into an MDP and generates a corresponding PRISM model for verification.
4. **Verification** — PRISM evaluates the MDP, producing an induced Discrete-Time Markov Chain (DTMC) and verifying the probability of success for the scenario.
5. **Navigation** — Dijkstra’s algorithm is applied to the verified DTMC to identify an optimal path, which is then interpreted by an LLM to generate a human-readable strategy explanation for the disaster manager.

Each stage produces distinct artefacts including logs, models, and strategy reports, ensuring transparency, reproducibility, and interpretability throughout the verification process. This architecture emphasises transparency, ensuring that outputs remain verifiable and traceable throughout the formal verification workflow.

In addition to the pipeline implementation, this study also evaluates the performance of several LLMs in independently solving the same disaster resourcing scenarios. The prompt used for these evaluations is provided in Appendix B, and all source code, artefacts, and experimental data are available in Appendix A.

The remainder of this dissertation follows the structure of the proposed pipeline. *Related Work* (Section 2) reviews prior research in formal verification and LLM-assisted modelling. *The Parser* (Section 3) outlines the scenario parsing process and structured representation. *The Composer* (Section 4) defines the resourcing problem and describes model generation. *Verification* (Section 5) explains how the models are evaluated within PRISM. *The Navigator* (Section 6) details strategy extraction and optimal path identification. *Results and Discussion* (Section 7) presents the experimental findings, and *Conclusions and*

Future Work (Section 8) summarises the outcomes and directions for further research.

2 Related Work

This section provides a brief review of related research in resource management, formal verification, and large language models (LLMs). It is not intended to be exhaustive. A search of relevant keywords in Scopus identified 14 published and 10 preprint works from 2024 to 2026, most of which were ultimately unrelated to the specific goal of combining generative AI with formal verification for disaster resourcing. The works discussed here represent the most directly relevant prior research and foundations for this study.

Recent developments in large language models have introduced new possibilities for integrating natural language reasoning with formal systems. Xu *et al.* [3] provide an interdisciplinary review of LLM applications in disaster management, identifying four primary operational domains—detection, tracking, analysis, and action—where LLMs can enhance situational awareness and decision support. Their review notes that most existing research focuses on data extraction and summarisation rather than high-level decision making, and calls for frameworks that connect LLM reasoning with structured, verifiable decision processes. The present work responds directly to this gap by linking generative AI interpretation with formal verification using PRISM, enabling natural language interaction while maintaining mathematical auditability.

Parallel studies have investigated the reliability and reasoning limitations of advanced models. Shojaei *et al.* [4] assess “Large Reasoning Models” such as o3-mini, Claude 3.7 Sonnet, and DeepSeek-R1 across controlled puzzle environments, showing that both reasoning and non-reasoning models experience rapid accuracy collapse beyond moderate complexity. Models often fixate on early incorrect paths or exhibit overthinking, where correct intermediate reasoning is followed by contradictory conclusions. These findings indicate that even reasoning-augmented models cannot yet guarantee algorithmic consistency, supporting the need for external verification in decision-critical domains such as disaster management.

Foundational analyses of large language models help explain these reasoning limitations. LLMs function primarily as probabilistic sequence predictors, trained to estimate the next most-likely token rather than to perform symbolic or causal reasoning [5, 6]. Because of this, they often produce responses that appear coherent but are guided by surface-level linguistic associations rather than structured inference.

Concerns about reliability extend beyond reasoning behaviour. Vendrow *et al.* [7] argue that existing LLM benchmarks often obscure model failures due to label errors and dataset contamination. Their proposed “platinum benchmarks” demonstrate that even state-of-the-art models continue to fail on basic arithmetic and logical inference tasks, highlighting the need for more rigorous evaluation. Likewise, Lott [8] provides an ongoing empirical benchmark using

Model	Offline Test	Mensa Norway
Grok-4	121	134
Claude-4-Opus	120	122
Gemini Pro	116	135
GPT-5-Pro	115	143
GPT-5	115	129
GPT-5-Thinking	113	140
Claude-4-Sonnet	111	123
GPT-5-Pro-Vision	108	120
Gemini Pro (Vision)	102	93
Perplexity	97	97
GPT-5-Thinking-Vision	94	90
DeepSeek-R1	93	112

Table 1: Comparison of models tested using the Mensa Norway and a custom question set, performed 2025-10-20.

the Mensa Norway and a custom question set (Table 1), offering an independent measure of model consistency under controlled, offline conditions. Together, these studies show that while generative models excel in linguistic fluency, their reasoning and verification reliability remain inconsistent.

A closely related preprint, VeriPlan, integrates large language models with formal verification for end-user planning tasks [9]. The system combines an LLM-based planner with a rule translator that converts natural language constraints into temporal logic, a set of user-adjustable flexibility controls, and a model checker that validates the generated plan against user-specified constraints. In a user study ($n = 12$), VeriPlan’s integration of formal verification improved perceived reliability, usefulness, and satisfaction, with the model checker functioning as a type of safety net to detect logical conflicts and provide corrective feedback. Although VeriPlan employs PRISM and Stormpy for model checking and shares this project’s emphasis on transparency and user accessibility, it focuses on generic task planning rather than disaster-resourcing. In contrast, the present work applies a similar synthesis of LLM reasoning and formal verification to a high-stakes, domain-specific context, automatically generating and verifying probabilistic resource distribution models rather than validating existing LLM-produced plans.

This dissertation builds upon these findings by combining LLM-driven model generation with formal verification to address these reliability gaps. By separating natural language interpretation from mathematical verification, the proposed PRISM pipeline ensures that even if an LLM produces inconsistent reasoning, the final verified outcome remains transparent, reproducible, and formally correct.

3 Parser

3.1 Scenario Parsing

The pipeline accepts user input describing a disaster scenario in natural language, including locations, routes, resource quantities, team capacities, and objectives. This input is processed by an AI model that extracts entities and relationships and converts them into a structured intermediate representation in JSON format. This stage exists to support reproducibility, transparency, and reliability: it produces an auditable artefact that records exactly how the system interpreted the input, and it standardises the information for use in later stages of the pipeline.

Pydantic is used to define the JSON schema and validate all model outputs, following OpenAI’s recommended approach to structured generation [10]. The AI model receives explicit schema constraints and is required to conform to them. Each pipeline instance automatically generates a unique directory containing the resulting JSON file and a log of the parsing process, ensuring traceability and facilitating later review.

3.2 Structured Representation with JSON Artefact

The JSON artefact defines the graph structure of the scenario, including nodes, edges, and resource attributes. Each edge encodes properties such as distance, safety classification, and capacity. Using JSON as an intermediate representation ensures consistency between natural language input and the mathematical model generated in the next stage. It also enhances transparency by enabling human auditing of both data and model assumptions.

In testing, this stage of the pipeline proved to be very reliable for the scenarios evaluated. For a given input, the resulting JSON was identical across dozens of runs, demonstrating deterministic reproducibility. While identical output does not guarantee correctness, the artefact is compact enough to be manually audited, and each scenario’s JSON file was verified to match the intended specification. The consistency and transparency of this stage make it a trusted foundation for the later steps. All generated JSON artefacts and their corresponding logs are available in Appendix A for verification and replication.

Although the final version of the pipeline uses ChatGPT 5 mini (pinned to `gpt-5-mini-2025-08-07`), experiments with the full ChatGPT 5 (pinned to `gpt-5-2025-08-07`) model produced identical results. The smaller model was therefore selected for efficiency without any observed loss in accuracy or reliability.

4 Composer

4.1 Model Generation

The JSON schema is transformed into a PRISM model file (`.prism`) through templated generation. Each node and route is encoded as part of a Markov Decision Process (MDP), capturing stochastic transitions that reflect real-world uncertainty. This stage produces both the model file and an associated property file (`.props`).

This stage uses ChatGPT 5 Mini (version 2025-08-07), which was selected after extensive testing for cost, reliability, and output quality. Earlier experiments using the standard ChatGPT 5 model, with default medium reasoning, frequently failed due to timeouts and were also prohibitively expensive: each pipeline run averaged over \$0.60 USD, regardless of success. The latest ChatGPT 4.1 model was also tested, but the 30,000-token limit on the research account was insufficient for generating complete PRISM models. ChatGPT 5 Mini provided comparable reliability at a significantly lower cost and much faster speed.

Occasional issues still occur, most commonly syntax errors in the generated PRISM file. To handle these, the pipeline includes a built-in error recovery mechanism offering three user options: (1) have the LLM attempt an automatic repair using the error message returned by PRISM, (2) have the LLM regenerate the model entirely, or (3) terminate the run. A *run* here refers to one complete execution of the pipeline, from natural language input through to verified output.

Another limitation is that PRISM models can become large and cumbersome to audit manually, with line counts varying substantially between runs. For the evaluated test scenario, model sizes ranged from 294 to 338 lines of code. Although PRISM ensures the correctness of the resulting verification, the generated model itself is not independently validated. To increase confidence, the pipeline records multiple artefacts for cross-checking: the optimal path and probability of satisfaction extracted from PRISM can be compared across runs, and the number of generated states can be validated against the theoretical state count for the given scenario.

These mechanisms provide a high level of trust in the system’s outputs. While the exact implementation of the PRISM model may differ slightly between runs due to LLM variability, the results remain reproducible and verifiable through the recorded artefacts.

4.2 The Resourcing Problem

This work aims to make formal verification for disaster resourcing usable by non-specialists without sacrificing rigour. We investigate whether generative AI can help a disaster manager move from a natural language description of a situation to a PRISM-verified strategy for resource distribution. The goal is a workflow that preserves transparency and reproducibility while reducing the

effort and expertise required to create correct models.

We formalise the scenario in terms of states, actions, and objectives that later map to an MDP for verification in PRISM. The next subsections specify the algebraic structures used to represent locations, resources, teams, and movement, and introduces the correctness condition used throughout this dissertation.

4.2.1 Algebraic Specification of Resource Distribution

Formal verification in this context is achieved using an MDP evaluated with the PRISM model checker. Before the verification process can be described in detail, it is necessary to define the algebraic structures that capture the state of the disaster scenario and its evolution through team actions. These definitions provide the foundation for constructing the formal model used by PRISM.

Definition 1 (Resourcing State) *The resourcing state is defined as the tuple $\sigma = (G, K, \alpha, \rho)$, such that $G = (V, E)$ is a graph representing the disaster-affected region, K is a logical predicate, $\alpha : X \rightarrow \mathbb{N}$ is the resource assignment function, and $\rho : T \rightarrow V$ is the team location function.*

Let *State* denote the set of all possible resourcing states. To model how teams change this state by moving resources between locations, we define a transition function as follows.

Definition 2 (Move Operation) *Let $move_i : State \times V \times V \times \mathbb{N} \rightarrow State$ be an operation for team $t_i \in T$, such that $\sigma' = move_i(\sigma, u, v, k)$ yields the updated state σ' after t_i moves k units of resource from location u to location v .*

Definitions 1 and 2 together specify the formal basis for the disaster resourcing problem. Each resourcing state σ encodes the full configuration of teams and resources, while each $move_i$ describes an atomic action that changes that configuration. These components later form the state and transition structure of the MDP evaluated in PRISM.

4.2.2 Case Study Scenario

To illustrate the formal definitions introduced in Section 4.2.1, we apply them to a concrete resourcing scenario used throughout this dissertation.

Figure 2 shows the locations of several Civil Defence Centres (CDCs) [11] in Auckland’s North Shore region. These sites are, at the time of writing, the active locations designated as safe areas during an emergency. Superimposed on these locations is the graph $G = (V, E)$, where the vertices $V = \{a, b, \dots, g\}$ represent CDCs, and the edges E represent routes between them with associated weights.

To better reflect real-world disaster conditions, and consistent with related work [2], each edge is annotated with a colour denoting the condition of that route. This classification is expressed as $C = \{R: \text{Red}, Y: \text{Yellow}, G: \text{Green}\}$, corresponding respectively to low, medium, and high probabilities of successful traversal for a response team. In this scenario, some routes are damaged

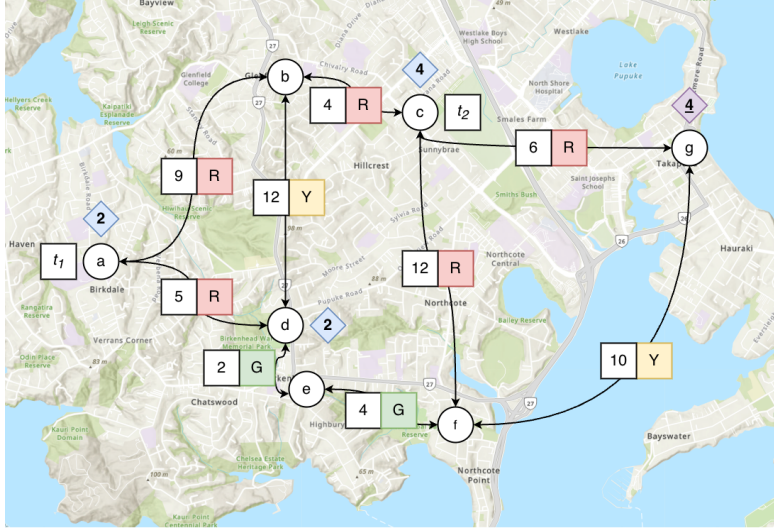


Figure 2: Auckland North Shore overlaid with graph G . Maps courtesy of ArcGIS [1].

or obstructed, meaning that teams may be unable to complete their journeys along those paths. While the example used here considers three distinct route conditions, the proposed framework allows disaster managers to refine the set C to suit the characteristics of any given disaster scenario.

4.2.3 Resource Allocation and Constraints

Following the scenario in Figure 2, we define how resources are represented, allocated, and constrained across the graph.

Adopting the notation of [12, 13, 2], let the set $X = \{ax, bx, \dots, gx\}$ represent the number of available resources at each vertex. Let $R = \{r_1, r_2, \dots, r_m\}$ denote the set of resource types, such as food, water, or medical supplies. For every vertex $v \in V$, there exists a corresponding variable $vx \in X$ representing the resource level associated with that location.

The *resource allocation function* α assigns quantities of resources to each location in the graph $G = (V, E)$. Formally,

$$\alpha : X \rightarrow \mathbb{N}^m$$

maps each variable $vx \in X$ to a tuple (a_1, a_2, \dots, a_m) , where each element corresponds to the quantity of a specific resource type $r_i \in R$ available at vertex v . In the simplified case where only one resource type is considered ($m = 1$), the mapping reduces to $\alpha : X \rightarrow \mathbb{N}$. The expression $\alpha(vx) = k$ indicates that location v currently stores k units of resource. This function provides the formal link between the graph representation of the disaster region and its quantitative resource state.

High-level resource requirements are expressed as logical predicates over the variables in X . Let K denote the conjunction of predicates that specify supply and demand conditions for resources across the graph. K is satisfiable if there exists an assignment function $\alpha : X \rightarrow \mathbb{N}$ such that all predicates in K evaluate to true, written as $K \models \alpha$.

To illustrate, consider the following example:

- Locations b and d each have at least two resources.
- Locations b and d have the same amount of resources.
- Location c has at least four resources.
- The number of resources at c is at most the sum of those at b and d .

Combining these predicates yields the complete resource requirement specification:

$$K \equiv (bx \geq 2) \wedge (dx \geq 2) \wedge (bx = dx) \wedge (cx \geq 4) \wedge (cx \leq bx + dx)$$

As shown in Figure 2, setting $\alpha(bx) = 2$, $\alpha(dx) = 2$, and $\alpha(cx) = 4$ satisfies all predicates, while all other vertices remain empty. In this case, $K \models \alpha$.

4.2.4 Teams and Movement Operations

In the *resourcing problem*, we are concerned not only with the initial distribution of resources in G , but also with how to develop a strategy for moving them optimally across the network. To represent this, we introduce the set of first responder teams responsible for transporting resources between locations.

Let $T = \{t_1, t_2, \dots, t_n\}$ denote the set of response teams. Each team $t_i \in T$ can occupy exactly one vertex $v \in V$ at a time. The *team location function* is defined as

$$\rho : T \rightarrow V,$$

where $\rho(t_i) = v$ indicates that team t_i is currently positioned at location v .

Building on Definition 1, the tuple $\sigma = (G, K, \alpha, \rho)$ represents a complete snapshot of the disaster environment at a given point in time. It encodes both the static structure of the region and its dynamic state: the topology of the network G , the logical goal conditions K , the current allocation of resources α , and the positions of response teams ρ . Together, these components form the foundation for describing system evolution through movement operations (Definition 2) and, ultimately, for constructing and verifying strategies in PRISM.

The movement of resources between locations is governed by the operation move_i defined in Definition 2. Given a current state σ , a move $\text{move}_i(\sigma, u, v, k)$ produces a new state σ' if and only if the following conditions are satisfied:

1. $\rho(t_i) = u$, meaning team t_i is currently located at u ;
2. $(u, v) \in E$, meaning a valid route exists between u and v ;

3. $\alpha(u) \geq k$, meaning at least k units of resource are available at u .

If all three conditions hold, the resulting state σ' is obtained by updating the allocation and team functions as follows:

$$\alpha'(u) = \alpha(u) - k, \quad \alpha'(v) = \alpha(v) + k, \quad \rho'(t_i) = v.$$

These updates represent the removal of k resources from u , their delivery to v , and the relocation of team t_i . All other components of σ remain unchanged, yielding $\text{move}_i(\sigma, u, v, k) = \sigma'$.

If any of the above conditions are not met, the move is invalid and the state remains unchanged:

$$\text{move}_i(\sigma, u, v, k) = \sigma.$$

Finally, since α may be zero, a move operation representing team movement without resource transfer can be written as

$$\text{move}_i(\sigma, u, v, 0).$$

4.2.5 Example Strategy and Risk Analysis

A complete resource distribution strategy, denoted by π , can be described as a finite sequence of valid move_i operations. A strategy π is considered *correct* if it results in a state σ' whose allocation function α' satisfies the demand requirement, expressed as $K \models \alpha'$.

Team	Route	CDist.	#Res.	Allocation α
t_1	$a \rightarrow d$	5	2	$g_x = 0$
t_1	$d \rightarrow e \rightarrow f \rightarrow g$	16	4	$g_x = 4$
t_2	$c \rightarrow g$	6	4	$g_x = 8$

Table 2: A correct strategy for K' .

A correct strategy for the scenario in Figure 2 is shown in Table 2. In this scenario, team t_1 begins at location a and team t_2 begins at c . The initial distribution of resources in G is two units each at a and d , and four units at c . The demand in this scenario, represented by an underline at location g , can be expressed as:

$$K' \equiv (g_x \geq 7).$$

The objective is to find a strategy π that concludes in a state σ' such that $K' \models \alpha'$.

Table 2 outlines a simple strategy in which t_1 moves along the route $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, collecting the two resources at a and, along the way, the two resources at d , before delivering all four to g . Meanwhile, t_2 transports the four resources at c directly along $c \rightarrow g$, delivering another four to g . Together, the eight resources delivered to g yield an α' that satisfies K' .

Although this strategy appears to be the most direct solution, several of the edges traversed are labelled *R*: Red, representing a low probability of successful traversal. If either t_1 or t_2 fails to complete its route, all resources it carries are lost. This concentration of risk creates a potential failure condition: if a team fails, the scenario becomes unsolvable because no α' can satisfy K' . In practice, it may be preferable for teams to carry fewer resources per traversal, introducing redundancy that increases the likelihood of eventually satisfying K' despite failures. Exploring when and why such trade-offs improve mission reliability leads directly to the use of Markov Decision Processes (MDPs).

4.3 Reasoning About Team Safety

Now that the system state, movement operation, and strategy have been defined, we can formalise how uncertainty is introduced into the model. In disaster scenarios, team actions such as traversing damaged routes are inherently probabilistic: even if a path is selected optimally, success is not guaranteed. To represent this uncertainty, the resourcing problem is expressed as an MDP, which forms the basis for verification in PRISM.

In probabilistic model checking, the entity that resolves nondeterminism is traditionally referred to as an *adversary* or a *policy*. Following PRISM's current terminology [14], this work instead uses the term *strategy*, which more naturally describes the sequence of team decisions in the disaster resourcing context. Formally, a strategy defines how actions are selected based on the current state, mapping $\pi : S \rightarrow Act$.

When a specific strategy is applied to an MDP, all nondeterminism is resolved and the model reduces to a Discrete-Time Markov Chain (DTMC), which enables probabilistic analysis of team safety and overall mission success.

We adopt the standard definitions of MDPs and DTMCs from [15], consistent with established probabilistic model checking literature.

Definition 3 (Markov Decision Process (MDP) [15]) *A Markov Decision Process is a tuple $M = (S, s_0, Act, P, L)$ where:*

- S is a finite set of states,
- $s_0 \in S$ is the initial state,
- Act is a finite set of actions,
- $P : S \times Act \times S \rightarrow [0, 1]$ is a probabilistic transition function such that $\sum_{s' \in S} P(s, a, s') = 1$ for all $s \in S$ and $a \in Act$,
- $L : S \rightarrow 2^{AP}$ is a labelling function that assigns to each state a set of atomic propositions from AP that hold in that state.

At each state $s \in S$, a nondeterministic choice of action $a \in Act$ is made, and the next state s' is chosen probabilistically according to the distribution $P(s, a, \cdot)$. An MDP therefore combines decision-making with stochastic outcomes, making it suitable for modelling disaster resourcing scenarios involving both human-controlled actions and environmental uncertainty.

4.3.1 MDP of the Resource Distribution Model

The resource distribution scenario defined in Section 4.2 can now be instantiated as a Markov Decision Process as described in Definition 3. Each resourcing state $\sigma = (G, K, \alpha, \rho)$ corresponds to a state $s \in S$, and each valid movement operation move_i for a team $t_i \in T$ constitutes an available action $a \in \text{Act}$. The transition probabilities $P(s, a, s')$ are determined by the safety classification of the edge traversed, such as $p(R) = 0.2$, $p(Y) = 0.6$, and $p(G) = 0.9$, while the labelling function $L(s)$ marks states that satisfy resource or goal conditions ($K \models \alpha$ or $K' \models \alpha'$). The initial configuration of the graph, resource allocation, and team positions defines the initial state s_0 .

This mapping captures both decision and uncertainty: the choice of action reflects deliberate team decisions, while the probabilistic outcome models environmental risk. PRISM evaluates such models against temporal logic properties over all possible executions, for example:

$$P_{\max}=?[F(K')],$$

which asks for the maximum probability of eventually reaching a state where the goal condition K' holds.

Transition Example Consider a team t_2 located at c in state σ_c . If the team attempts to move from c to g along an edge classified as **RED**, the probability of successfully reaching the next state σ_g is equal to the traversal probability $p(\text{RED})$, and the probability of failure is $1 - p(\text{RED})$. Formally:

$$P(\sigma_c, \text{move}_2(\sigma_c, c, g, 4), \sigma_g) = p(\text{RED}), \quad P(\sigma_c, \text{move}_2(\sigma_c, c, g, 4), \sigma_{\perp}) = 1 - p(\text{RED}),$$

where σ_{\perp} represents the failure state. This construction captures the inherent uncertainty in route safety while maintaining a formally verifiable state-transition structure.

When a strategy π resolves the nondeterminism in this MDP, the resulting system forms an induced Discrete-Time Markov Chain (DTMC), which is then analysed to compute the probability of success or failure over all reachable states.

5 Verification

5.1 Verification and Strategy Extraction

The PRISM model checker evaluates the generated Markov Decision Process against the specified temporal logic properties. For the tested scenario, the primary verification goal was to determine the maximum probability of successfully delivering at least seven resources to location g , expressed as $P_{\max}=?[F(K')]$. This property queries the probability of eventually reaching a state in which the goal condition K' is satisfied.

The safest resource distribution strategy is obtained by verifying the Probabilistic Computation Tree Logic (PCTL) [15] formula $P_{\max}=?[F(K')]$, where

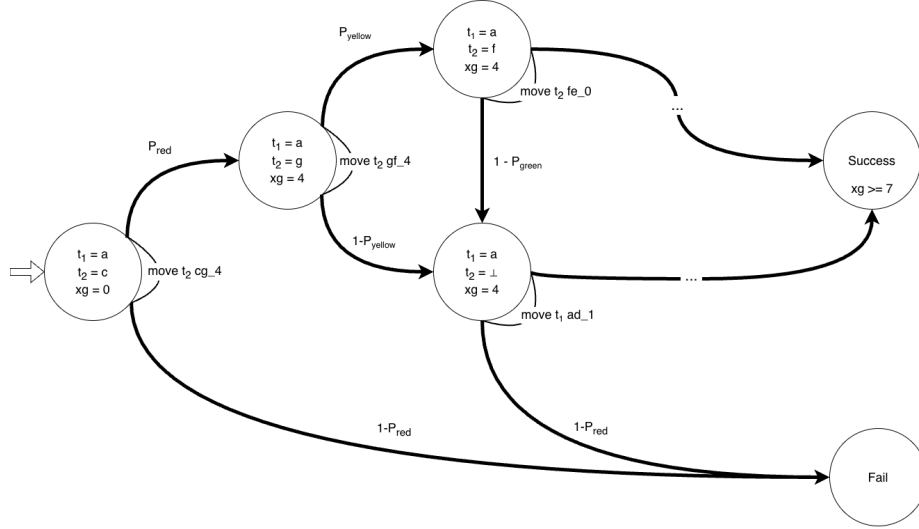


Figure 3: Truncated DTMC graph induced from the scenario in Figure 2, showing the optimal strategy obtained from verification of $P_{\max}=?[F(K')]$.

the P operator computes the strategy that maximises the probability of reaching a state σ' whose allocation function satisfies $\alpha' \models K'$. PCTL provides a formal framework for expressing and evaluating probabilistic reachability and safety properties over Markov models, as detailed in [15].

5.1.1 Discrete-Time Markov Chains (DTMCs)

A Discrete-Time Markov Chain (DTMC) is a special case of a Markov Decision Process (MDP) where each state has only a single available action, producing a single probabilistic transition outcome. Following the formal definition in [15]:

Definition 4 (Discrete-Time Markov Chain (DTMC) [15]) A (discrete-time) Markov chain is a tuple $M = (S, P, \iota_{init}, AP, L)$ where:

- S is a countable, nonempty set of states;
- $P : S \times S \rightarrow [0, 1]$ is the transition probability function such that, for all states s , $\sum_{s' \in S} P(s, s') = 1$;
- $\iota_{init} : S \rightarrow [0, 1]$ is the initial distribution such that $\sum_{s \in S} \iota_{init}(s) = 1$;
- AP is a set of atomic propositions; and
- $L : S \rightarrow 2^{AP}$ is a labelling function.

Any Markov chain can be viewed as an MDP in which the probability distribution in each state is uniquely determined [15]. A DTMC therefore arises

from an MDP when nondeterminism is resolved by fixing a single action for each state, typically through the optimal strategy that specifies the chosen action for every state. Once this strategy is applied, PRISM induces a deterministic transition structure that contains only probabilistic transitions, allowing analysis of reachability and success probabilities.

In the PRISM pipeline, resolving the nondeterminism in the MDP by applying the optimal strategy produces an induced DTMC that represents the probabilistic behaviour of that verified strategy. This structure enables computation of reachability probabilities such as $\text{Pmax}=?[F(K')]$, which expresses the likelihood of eventually reaching a state where the goal condition K' holds. Figure 3 shows a truncated portion of the induced DTMC for the scenario in Figure 2, where each node corresponds to a system state annotated with team locations and whether x_g satisfies the goal, and edges denote probabilistic transitions. Transitions that represent failed traversals terminate in the failure state, denoted by \perp .

5.1.2 Model Verification in PRISM

When the LLM-generated MDP is submitted to PRISM, the model checker first performs symbolic state-space construction and strategy synthesis, resolving nondeterministic actions according to the optimal strategy for the specified property $\text{Pmax}=?[F(K')]$. During this initial verification phase, PRISM reports the total number of states, transitions, and the probability of satisfying the goal condition. Across all test runs, these three values were identical, supporting the claim that the MDPs generated by the Composer stage are semantically equivalent and that the verification process is fully reproducible.

The first verification step produces the files `strat.all`, `strat.lab`, `strat.sta`, and `strat.tra`, which together describe the complete optimal strategy and its corresponding probabilistic structure. The pipeline then re-submits `strat.all` to PRISM to generate a restricted version of the model containing only the transitions and states reachable under that verified optimal strategy. This produces the reduced files `restricted.tra`, `restricted.sta`, and `restricted.lab`, which together form the induced DTMC representing the verified behaviour of the system.

Each of these artefacts serves a specific analytical purpose:

- `.sta` — records the full state vector for each DTMC node;
- `.lab` — identifies labelled states, including "init" and "goal";
- `.tra` — defines the probabilistic transitions between states.

The restricted DTMC is then processed using Dijkstra’s algorithm to extract the most probable path from the initial to the goal state. This step is described in detail in Section 6.

This two-phase process confines PRISM’s role to formal verification and the export of verifiable artefacts, leaving interpretation and path reconstruction to

later stages of the pipeline. Maintaining this separation between composition, verification, and navigation supports reproducibility, transparency, and consistent comparison across all LLM-generated scenarios.

6 Navigation

The final stage of the pipeline interprets PRISM’s verified outputs to produce a human-readable strategy. Using the restricted transition (`.tra`), label (`.lab`), and state (`.sta`) files exported during verification, this stage reconstructs the induced DTMC and applies Dijkstra’s algorithm [16] to identify the most probable path from the initial to the goal state. The resulting path is written to `optimal_path.txt`, which lists the ordered sequence of states along the optimal route. This file was identical across all verified runs, supporting the consistency of the underlying verification process.

The `optimal_path.txt` file is then passed to a large language model, ChatGPT 5 Mini (pinned to version `gpt-5-mini-2025-08-07`). The LLM then generates a natural language summary of the verified strategy which is saved as `strategy_explanation.md`. This step aims to make the verified results more accessible to disaster managers by describing the optimal strategy in clear, domain-relevant language. However, this stage proved unreliable: the model occasionally omitted or misinterpreted steps, leading to incomplete or inaccurate explanations. Future work should explore alternative models, prompting strategies, or structured output formats to improve the consistency and correctness of this interpretive stage.

The artefacts generated during navigation complete the end-to-end verification workflow. Together, these outputs enable both quantitative evaluation and qualitative interpretation of the verified strategies, as discussed in Section 7. As with every other step in the pipeline, implementation details and data are available in Appendix A.

7 Results and Discussion

7.1 Overview

Table 3 compares three full runs of the complete PRISM pipeline with direct evaluations of the same scenario using a range of large language models. Each model received the same evaluation prompt provided in Appendix B, while the PRISM pipeline followed the automated workflow described in this study.

The models were chosen to roughly align with those in Table 1, excluding vision-enabled variants since this task has no visual component. Some substitutions were made for availability and cost reasons. DeepSeek V3.2 with DeepThink enabled was used in place of DeepSeek-R1, which was unavailable. GPT-5 Pro was also tested once through the API but timed out after approximately 30 minutes, incurring a charge of over \$7 USD without producing output.

	Run 1	Run 2	Run 3
PRISM Pipeline	0.255533	0.255533	0.255533
Grok 4 Expert Mode	0.208	0.20827125	0.2083
Claude Opus 4.1 Thinking	0.208	0.208	0.2083
Gemini 2.5 Pro	0.104	0.2084	0.2084
ChatGPT-5	0.2083	0.290444	0.20827125
ChatGPT-5 Mini	0.243235	0.20827	0.20827125
Claude Sonnet 4.5 Thinking	0.208	0.2074	0.207
Claude Sonnet 4.5	0.2079	0.2081	0.1474
Perplexity Sonar	0.2083	0.20827125	0.0002
DeepSeek V3.2 DeepThink	0.2083	0.2083	0.2083

Table 3: Probability of success outputs from various models across three runs.

Since GPT-5 Mini is used exclusively in the PRISM pipeline and provides a comparable reasoning architecture at much lower cost, it was included as a separate and directly relevant comparison.

To ensure that all models were evaluated fairly, each was given the same detailed scenario and contextual prompt:

You are an expert in logistics and supply chain management, specialising in optimising resource allocation and routing under constraints. You have a deep understanding of graph theory, probability, and optimisation techniques.

This prompt clearly defines the task, the available resources and teams, the safety of each route, and the overall objective, allowing a direct comparison between free-form LLM reasoning and the PRISM-verified results.

Across most models, the reported probabilities were inconsistent between runs. While several converged near 0.208, none except DeepSeek V3.2 produced the same result three times in a row. DeepSeek V3.2 was the only model to yield identical numeric outputs across all runs, though its output matched the same baseline strategy as most other models rather than a distinct or improved solution. In other cases, the same model generated noticeably different plans for the same scenario, sometimes introducing missing or illogical movements. These inconsistencies illustrate how even small variations in reasoning can lead to divergent outcomes.

By contrast, the PRISM pipeline produced identical results in every execution, with a verified success probability of 0.255533 and a consistent optimal strategy. This repeatability supports the validity of the LLM-generated models passed into PRISM and highlights the difference between approximate reasoning and verifiable computation. While generative AI systems offer a far more accessible interface, they lack the reliability required for safety-critical decision making. A discrepancy of only a few percentage points in probability could represent a substantial change in expected outcomes during a real disaster response.

7.2 Trying to Solve with AI Directly

7.2.1 Overview

To contrast the PRISM pipeline with unverified AI reasoning, several large language models were tested on the same disaster resourcing scenario without access to formal verification. Each model received the identical evaluation prompt provided in Appendix B, and all runs were performed in fresh chat sessions to avoid caching effects. Full transcripts are archived in the repository referenced in Appendix A.

Across models, most correctly reproduced the structure of the scenario and proposed strategies that initially appeared sound. However, probabilistic reasoning was frequently unreliable. Common errors included incorrect probability multiplication, logically inconsistent contingencies, and violations of basic scenario rules, such as assuming that resources carried by failed teams could still be recovered. While the models often produced persuasive and well-structured explanations, their outputs were at times mathematically incorrect or internally inconsistent.

This comparison highlights a key advantage of the PRISM-based pipeline: whereas standalone LLMs can only approximate reasoning about uncertainty, the integrated approach ensures verifiable correctness through formal probabilistic model checking.

7.2.2 Grok-4

Metric	Value	Notes
Distinct plans	1	Same route each run
Reported P_{success}	0.2080, 0.20827125, 0.2083	Rounding differences
Contingency planning	No	Single-shot plan

Table 4: Run metrics and strategy summary for Grok 4.

Setup Grok 4, with Expert mode enabled, was accessed through the web interface [17].

All Runs As all runs produced near-identical outputs, they are discussed collectively here. Each of the three attempts identified the same strategy: Team 1 collected all resources at a , moved to d to collect the remaining resources for a total of four, and then proceeded along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, delivering all four resources to g . Team 2 collected all four resources from c and moved directly along $c \rightarrow g$, also delivering four resources. The resulting total was eight resources at g .

The probability of success was correctly computed in all cases, although each run reported a slightly different level of precision: 0.208, 0.20827125, and

0.2083. There is no clear reason for this variation, but all three calculations are consistent in methodology. Specifically, Team 1’s path had a success probability of $0.5 \times 0.99 \times 0.99 \times 0.85 = 0.4165$, and Team 2’s path 0.5, giving a combined probability of $P_{\text{success}} = 0.4165 \times 0.5 = 0.2083$.

None of the Grok 4 runs produced contingency strategies or alternative routes. Each solution approached the problem in the most direct manner, treating the teams’ strategies as entirely independent. Consequently, failure by either team results in complete failure of the scenario, as no backup actions were generated to preserve resource delivery.

7.2.3 Claude 4.1 Opus

Metric	Value	Notes
Distinct plans	1	Same route each run
Reported P_{success}	0.208, 0.208, 0.2083	Rounding differences only
Contingency planning	No	Single-shot plan

Table 5: Run metrics and strategy summary for Claude 4.1 Opus.

Setup Claude 4.1 Opus (`claude-opus-4-1-20250805-thinking-16k`) was accessed through the LMArena.ai interface [18].

All Runs Across three independent runs, Claude 4.1 Opus produced outputs that were nearly identical in structure and reasoning to those of Grok 4. Both teams followed the same routes, and the model did not propose any variations or contingency strategies. Team 1 collected both resources at a , proceeded to d to gather the supplies there, and then continued along the path $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, delivering four resources to g . Team 2 moved directly along $c \rightarrow g$, also delivering four resources.

The probability of success for Team 1 was calculated as $0.5 \times 0.99 \times 0.99 \times 0.85 = 0.4165$, and for Team 2 as 0.5. The combined probability of both teams successfully completing their routes was therefore $P_{\text{success}} = 0.4165 \times 0.5 = 0.2083$.

The reported probabilities of success across runs were consistent: 0.208, 0.208, and 0.2083. In terms of output precision, Claude 4.1 Opus was the most consistent of all evaluated models, with only a single decimal place of variation observed across runs. As with the Grok 4 outputs, these small differences reflect rounding rather than divergent reasoning. No alternate routes, fallback actions, or adaptive behaviours were generated, and both teams’ strategies were treated as independent and non-coordinated.

7.2.4 Gemini 2.5 Pro

Setup Gemini 2.5 Pro was accessed through Google’s AI Studio interface [19].

Metric	Value	Notes
Distinct plans	2	Run 1 used a unique multi-pass route
Reported P_{success}	0.104, 0.2084, 0.2084	Run 1 had a distinct plan
Contingency planning	No	Single-shot plan

Table 6: Run metrics and strategy summary for Gemini 2.5 Pro.

Run 1 In Run 1, Gemini 2.5 Pro produced an unconventional and inefficient strategy that diverged from the consistent approaches seen in other models. Team 1, starting at a , ignored the resources available there and first travelled from a to d (Red route) to collect two resources. It then continued along $d \rightarrow e \rightarrow f \rightarrow g$, delivering the two resources to g . After this delivery, Team 1 returned to a via $g \rightarrow f \rightarrow e \rightarrow d \rightarrow a$, collected the remaining two resources, and repeated the route $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ to deliver them. Gemini justified this repetition by stating:

A second trip is necessary as T1 cannot carry all four resources from 'a' and 'd' while following the safest path without backtracking.

This reasoning is puzzling because it is unnecessary for Team 1 to carry four resources from a to d , as two resources already begin at d .

Team 2, starting at c , collected four resources and travelled directly along the Red route $c \rightarrow g$ to deliver them, consistent with the other models' results. Gemini reported a success probability of $P_{\text{success}} = 0.104$, though no calculation steps were provided. This is inaccurate as, assuming independent traversal probabilities, Team 1's single-route probability is 0.4165. Because the model required three traversals, this yields $0.4165^3 = 0.0722$, which, when multiplied by Team 2's 0.5 success probability, gives an overall estimate of $P_{\text{success}} = 0.0361$. As Gemini's 0.104 is exactly half of the 0.208 that most models ended up with for their paths, Gemini seems to have multiplied the probabilities for $a \rightarrow \dots \rightarrow g$ and $c \rightarrow g$ by 0.5.

Gemini further stated:

No feasible plan with a higher probability of success exists.

This was contradicted by the other two runs that Gemini conducted.

While this plan superficially resembles the PRISM-verified path, it is in fact significantly worse: the only part of the path where it could be argued there is redundancy is Team 1's initial trip from $a \rightarrow d$, because it does not have any resources, and Team 1's trip from $g \rightarrow \dots \rightarrow a$, for the same reason. In theory, Team 2 could recover if Team 1 failed during those parts of the plan. This is generous, Gemini did not suggest this was possible.

In short, this run produced an unsafe, inefficient path, and gave a significantly overstated probability of success along with it.

Runs 2 and 3 Runs 2 and 3 produced outputs nearly identical in structure and reasoning to those of Grok 4 and Claude 4.1 Opus. Both teams followed the same routes, and the model did not propose any alternative or contingency strategies. Team 1 collected both resources at a , proceeded to d to gather the supplies there, and then travelled along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, delivering four resources to g . Team 2 moved directly along $c \rightarrow g$, also delivering four resources. Together, both teams completed the delivery of eight total resources to g .

The probabilities of success were consistent across both runs, reported as 0.20827125 and 0.2083. These values align closely with those obtained from Grok 4 and Claude 4.1 Opus, differing only in output rounding. No adaptive or alternate behaviours were generated, and both teams’ strategies were treated as independent and non-coordinated.

7.2.5 ChatGPT 5

Metric	Value	Notes
Distinct plans	2	Contingency in Run 2
Reported P_{success}	0.2083, 0.290444, 0.20827125	Run 2 was distinct
Contingency planning	Yes	Only for Run 2

Table 7: Run metrics and strategy summary for ChatGPT 5.

Setup ChatGPT 5 (gpt-5-2025-08-07) was accessed through the OpenAI API using the “medium thinking” configuration, which is the default.

Runs 1 and 3 Runs 1 and 3 produced results that matched the other models studied so far. Team 1 collected resources at a , travelled to d to gather additional supplies, then proceeded along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, delivering four resources to g . Team 2 moved directly along $c \rightarrow g$, delivering four resources. The reported probabilities of success were 0.2083 and 0.20827125, aligning with prior models except for rounding differences. No alternative routes or contingency actions were proposed.

Run 2 Here, finally, we have an attempt at creating a path with redundancy for failure recovery. While this alternate path is far from PRISM’s verified optimal solution, it is the most sophisticated plan produced so far. In this run, ChatGPT 5 assigned Team 2 to follow the usual route $c \rightarrow g$ with all four resources. Team 1 collected one resource at a , then picked up two at d and travelled $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$. If Team 1 failed on the $a \rightarrow d$ segment, Team 2 could still attempt to recover the remaining resources.

The proposed contingency has Team 2 travel $g \rightarrow f \rightarrow e \rightarrow d$ to collect the two resources at d and return to g , then, if successful, travel $g \rightarrow f \rightarrow e \rightarrow d \rightarrow a$

to collect the last resource and return to g . This creates contingency only for the $a \rightarrow d$ crossing, and the recovery sequence is inefficient. Any failure before all three remaining resources reach g results in total failure. Avoiding the initial red edge to a offers no benefit, since Team 2 must traverse it later, and in this plan $g \rightarrow d$ is traversed more than necessary. Nonetheless, this is the first plan with explicit contingency behaviour.

ChatGPT 5 reported a probability of success $P_{\text{success}} = 0.290444$, with the following breakdown:

$$\begin{aligned}
&\text{Standard path: } 0.5 \times 0.99 \times 0.99 \times 0.85 \times 0.5 = 0.2082713, \\
&\text{T1 main success} = 0.4165425, \\
&\text{T2 main success} = 0.5, \\
&\text{T1 } g \leftrightarrow c \text{ salvage} = 0.25, \\
&\text{T2 } g \leftrightarrow d \text{ for 2 units} = 0.6940306, \\
&\text{T2 } g \leftrightarrow a \text{ for 1 unit} = 0.1735077, \\
&\text{T1 recovers, T2 fails: } 0.4165425 \times 0.5 \times 0.25 = 0.0520678, \\
&\text{T2 recovers, T1 fails early: } 0.5 \times 0.5 \times 0.6940306 \times 0.1735077 = 0.0301049, \\
&\text{Total: } 0.2082713 + 0.0520678 + 0.0301049 = 0.290444.
\end{aligned}$$

This calculation is incorrect. Team 1 cannot recover Team 2 in the described plan, since Team 2 carries all four resources from c to g in one trip. If Team 2 fails, the mission fails. Removing the “T1 recovers, T2 fails” component yields $P_{\text{success}} \approx 0.2383762$, which would still be the second-highest self-reported probability among the models considered, after the initial ChatGPT 5 Mini run.

In summary, ChatGPT 5 was the first model to produce a plan with redundancy. Although the approach was inefficient, the reported probability was incorrect, and some scenario constraints were ignored, it did demonstrate a basic notion of contingency. This reinforces a broader concern for disaster management: large language models often generate plausible but unreliable plans. Every model evaluated here stated that it had found an optimal plan; in each case, this claim was false.

7.2.6 ChatGPT 5 Mini

Metric	Value	Notes
Distinct plans	2	Run 1 differs
Reported P_{success}	0.243235, 0.20827, 0.20827125	Run 1 distinct
Contingency planning	Claimed	Unsubstantiated

Table 8: Run metrics and strategy summary for ChatGPT 5 Mini.

Setup ChatGPT 5 Mini (`gpt-5-mini-2025-08-07`) was accessed through the OpenAI API using the default configuration.

Run 1 Run 1 claims a higher probability of success, making ChatGPT 5 Mini the highest self-reported result at 0.243235. It attributes this to a plan it describes as having contingencies. The plan is as follows: Team 2 first moves $c \rightarrow g$ with all four resources. If that succeeds, Team 2 then travels $g \rightarrow f \rightarrow e \rightarrow d$, picks up both resources at d , and returns to g . If all of that also succeeds, Team 1, still at a , traverses $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ carrying the two resources from a . If Team 2 fails on $g \rightarrow d$, Team 1 takes the same route but also picks up the resources at d .

The calculations used to reach $P_{\text{success}} \approx 0.243235$ rely on the assumption that after completing $c \rightarrow g$, then gone $g \leftrightarrow d$, Team 2 can recover if Team 1 fails at its path from $a \rightarrow g$. But the plan has Team 1 taking both resources from a , so Team 1 failing leaves, at most, 6 resources available, resulting in a fail state. The model calculates that having a team travel $g \leftrightarrow d$ and another team travel $a \rightarrow g$, where either team failing is a total failure, is safer than having a team travel $a \rightarrow g$.

To approximate the actual probability for this strategy previously established values are used. There is 0.5 from Team 2's $c \rightarrow g$ path, this is multiplied by the 0.4165425 from Team 1's $a \rightarrow g$, as that also has to succeed to satisfy this scenario. Then there is Team 2's $g \leftrightarrow d$, which only affects the probability if the first leg $g \rightarrow d$ succeeds, in which case its value 0.833085 is multiplied in. Thus, If Team 2 completes the first leg successfully, $P_{\text{success}} \approx 0.1735076$; if it fails, $P_{\text{success}} \approx 0.2082712$. If Team 2 does not attempt $g \leftrightarrow d$ at all, P_{success} remains approximately 0.2082712. The staged plan therefore does not create redundancy; it introduces extra failure opportunities while preserving the same critical edges. Its claimed advantage rests on counting recoveries that violate the rule that if a team that fails is carrying resources, the resources are lost.

This makes Run 1 one of the weakest results: the explanation is persuasive, the arithmetic appears rigorous, yet the assumptions are inconsistent with the scenario rules and the structure adds avoidable risk. It is weaker than ChatGPT 5's Run 2, which at least introduces a genuine conditional recovery point, even though its probability arithmetic was incorrect.

Runs 2 and 3 Runs 2 and 3 followed the standard strategy observed across most other models. Team 1 collected resources at a , moved to d to gather additional supplies, and then travelled along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, delivering four resources to g . Team 2 transported four resources along $c \rightarrow g$. The reported success probabilities were 0.20827 and 0.20827125, values consistent with the typical range found in the other models, the only difference being the rounding. No contingency routes were proposed in these runs, although the model specifically mentioned that it had explored such plans and found them to have lower overall reliability, an unsubstantiated claim that is inconsistent with both the PRISM-verified analysis and ChatGPT 5's conditional plan.

Metric	Value	Notes
Distinct plans	1	Same route each run
Reported P_{success}	0.208, 0.2074, 0.207	Minor maths errors
Contingency planning	No	Single-shot plan

Table 9: Run metrics and strategy summary for Claude Sonnet 4.5 Thinking.

7.2.7 Claude Sonnet 4.5 Thinking

Setup Claude Sonnet 4.5 Thinking was accessed through Claude’s web interface [20], with *Extended thinking* enabled.

All Runs All three runs followed the standard plan used by other models. Team 1 travelled $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ after collecting at a and d . Run 2 deviated slightly, only picked up 1 resource at d , but this makes no practical difference for this scenario. Team 2 travelled $c \rightarrow g$ with four resources. Reported success probabilities were 0.208, 0.2074, and 0.207.

Runs 2 and 3 were slightly incorrect due to arithmetic. For example, Run 2 computed

$$0.5 \times 0.99 \times 0.99 \times 0.85 = 0.4148,$$

but the correct product is

$$0.5 \times 0.99 \times 0.99 \times 0.85 = 0.4165.$$

This led to

$$0.4148 \times 0.5 = 0.2074$$

instead of the correct

$$0.4165 \times 0.5 = 0.2083.$$

The discrepancy is small, and the route choice matches other models, but the numerical error is curious given the otherwise identical setup and paths.

7.2.8 Claude Sonnet 4.5

Metric	Value	Notes
Distinct plans	2	Run 3 differs from Runs 1–2
Reported P_{success}	0.2079, 0.2081, 0.1474	Minor maths errors
Contingency planning	No	Single-shot plan

Table 10: Run metrics and strategy summary for Claude Sonnet 4.5.

Setup Claude Sonnet 4.5 was accessed through Claude’s web interface [20], with *Extended thinking* disabled.

Runs 1 and 2 Runs 1 and 2 both produced the standard plan observed in nearly all previous models. Team 1 collected the resources at a and d , then travelled along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ to deliver four resources to g . Team 2 transported four resources along $c \rightarrow g$. The reported probabilities of success were 0.2079 and 0.2081, values consistent with the expected baseline of approximately 0.2083.

In Run 2, the model initially considered several alternative plans before converging on the same configuration as most of the other systems. Minor arithmetic errors were again present, similar to those seen in the Claude Sonnet 4.5 Thinking model. Despite these minor discrepancies, both runs follow the standard non-contingent strategy.

Run 3 Run 3 diverged notably from the standard pattern. In this case, Claude Sonnet 4.5 identified the $d \rightarrow e \rightarrow f \rightarrow g$ corridor as a highly reliable path and attempted to exploit it by pooling resources at d . This produced an alternative strategy: Team 2 travelled $c \rightarrow b \rightarrow d$, then followed $d \rightarrow e \rightarrow f \rightarrow g$, while Team 1 moved directly along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$. The model calculated the probability for Team 2’s full path as $0.425 \times 0.8326 = 0.3539$, and for Team 1’s path as 0.4163, resulting in a combined probability of $0.4163 \times 0.3539 = 0.1474$.

This decision reflects a clear but misplaced heuristic: the model correctly recognised that $d \rightarrow e \rightarrow f \rightarrow g$ is a relatively safe sequence, but it failed to evaluate the trade-off introduced by replacing the direct $c \rightarrow g$ route (with $p = 0.5$) with a longer, less reliable alternative. The reasoning effectively overemphasised the safety of the green and yellow corridor, without considering that the additional traversal $c \rightarrow b \rightarrow d$ (with $p = 0.425$) reduced overall reliability. The resulting $P_{\text{success}} = 0.1474$ is mathematically consistent with the strategy chosen but demonstrates a flawed optimisation that prioritised local edge safety over total path reliability.

7.2.9 Perplexity Sonar

Metric	Value	Notes
Distinct plans	2	Run 3 uniquely low
Reported P_{success}	0.2083, 0.20827125, 0.0002	Run 3 very low
Contingency planning	No	Single-shot plan

Table 11: Run metrics and strategy summary for Perplexity Sonar.

Setup Perplexity Sonar was accessed through Perplexity’s web interface, using the default *Search* setting [21], with *Pro* enabled.

Runs 1 and 2 Runs 1 and 2 both followed the standard plan observed across most models. Team 1 collected the resources at a and d , then travelled along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ to deliver four resources to g . Team 2 transported four resources along $c \rightarrow g$. The reported success probabilities were 0.2083 and 0.20827125, consistent with the expected baseline of approximately 0.2083 and identical to the results of models such as Grok 4, Claude 4.1 Opus, Gemini 2.5 Pro, and ChatGPT 5.

Run 3 Run 3 produced the lowest reported probability of success of any model or run, with $P_{\text{success}} = 0.0002$. The model found the standard paths but then applied an incorrect probability formulation, using

$$P_{\text{all}} = (0.4165)^2 \times (0.8331)^2 \times (0.5)^4,$$

interpreting these as per-resource probabilities for the two resources from a , two from d , and four from c . It justified this approach as a deliberately conservative interpretation, claiming that modelling each resource independently would underestimate rather than overestimate reliability. The model stated:

This is the product of each resource’s probability of reaching g safely, assuming independent trials for each team and resource (which is conservative, since teams can fail with all their cargo).

In the described plan, Team 1 carries both a and d resources together along $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$, already incorporating the $d \rightarrow e \rightarrow f \rightarrow g$ segment ($p = 0.8331$) into its total route probability of 0.4165. Treating $d \rightarrow g$ as an independent term double-counts that segment.

Even accepting the erroneous per-resource independence assumption, the arithmetic itself is also incorrect. Evaluating the expression gives

$$(0.4165)^2 \times (0.8331)^2 \times (0.5)^4 \approx 0.007525,$$

not 0.0002. When queried about the discrepancy, the model recomputed the expression and blithely returned 0.0075 without any further reflection. It is unclear how it arrived at the same strategy as the other runs yet performed the calculation so incorrectly. Run 3 therefore represents a severe reasoning failure: Perplexity Sonar found the same paths as most other runs but applied an invalid per-resource probability model and a basic arithmetic error, resulting in by far the lowest reported value across all systems.

7.2.10 DeepSeek V3.2 (DeepThink Enabled)

Setup DeepSeek V3.2 was accessed through DeepSeek’s web interface [22], with the *DeepThink* mode enabled.

All Runs All three runs produced substantially different textual explanations but converged on the same underlying plan and probability calculations. Each

Metric	Value	Notes
Distinct plans	1	Same route all runs
Reported P_{success}	0.2083, 0.2083, 0.2083	Identical
Contingency planning	No	Single-shot plan

Table 12: Run metrics and strategy summary for DeepSeek V3.2 (DeepThink Enabled).

described the standard two-team configuration: Team 1 collected resources at a and d , then travelled $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g$ to deliver four resources to g , while Team 2 transported four resources directly along $c \rightarrow g$. The reported probabilities of success were 0.2083, 0.2083, and 0.2083, values that are consistent both with one another and with the baseline results obtained from other models. In fact, DeepSeek was the only model to return exactly the same probability across all runs. Although the explanations varied in structure, each produced the same correct calculation and strategy, with no alternative routes or contingency behaviour proposed.

7.3 PRISM Pipeline Results

7.3.1 Introduction

Run	P_{success}	Sim Steps	P_{path}	States	Transitions	LoC
1	0.255533	24	0.020477	241,992	241,992	294
2	0.255533	24	0.020477	241,992	241,992	338
3	0.255533	24	0.020477	241,992	241,992	300

Table 13: PRISM verification metrics for the three full pipeline runs. Actions are the given optimal actions at the end of the pipeline. All runs produced identical state and transition counts, demonstrating structural reproducibility.

Unlike the LLM runs reviewed above, the PRISM pipeline runs were entirely consistent. PRISM produced identical results across all runs: the total probability of success, the optimal-path probability, and the number of simulation steps were all the same. Both the simulator output and the optimal path generated using Dijkstra’s algorithm from the induced strategy file yielded identical sequences of actions and identical step counts. Formally, the optimal path corresponds to the model reaching a state where the modified resource requirement $K' \models \alpha'$, satisfying the condition $x_g \geq 7$. One of the three runs initially encountered a syntax error in the PRISM model generated by the composer module, but this was automatically corrected through the pipeline’s error recovery mechanism, which uses an LLM to repair or regenerate the model. All recovery events are logged in the run metadata, and the full code and outputs are available in Appendix A.

The *Strategy Explanation* artefact, generated by ChatGPT 5 Mini using the optimal path from PRISM, occasionally contained minor inconsistencies with the verified model. Because it is produced only after verification and does not influence any prior pipeline stages, these differences have no effect on the outcomes reported here. Future work may consider testing alternative LLMs or parameter settings to improve the quality of this final output. In all runs, the state and transition counts reported by PRISM were identical. The only notable variation was in the number of lines of code (LoC) in the generated models, which differed by more than 10%. This variation did not affect model semantics or results, as there are many syntactically distinct but logically equivalent ways to encode the same scenario in PRISM. The LLM also alternated between two syntactic formulations of the property to be verified, either $P_{\max}=?[F(xg>=7)]$ or $P_{\max}=?[F\text{"goal"}]$, where label "goal" = $(xg \geq 7)$. Both are equivalent under PRISM's semantics.

The probability of success, P_{success} , was identical in every run. After correcting for the arithmetic errors observed in the LLM-generated results, the PRISM-verified models achieved the highest success probability of any approach tested. This is expected, as PRISM formally explores the entire reachable state space and computes

$$P_{\max} = P[F K'],$$

representing the maximal probability of eventually reaching a state where the modified resource requirement K' is satisfied.

The distinction between the probability of a single path and the probability of success under a verified strategy is important. The reported optimal-path probability, $P_{\text{path}} = 0.020477$, is low because the path contains many steps and is designed with redundancy. It measures only the likelihood that this exact sequence executes without any failure. In contrast, the overall success probability, $P_{\text{success}} = 0.255533$, computed by PRISM, aggregates all successful executions that still satisfy K' (that is, reach a state with $x_g \geq 7$), including branches where early actions may fail but later ones recover. Redundancy reduces the probability of a single flawless traversal but increases the number of successful branches, resulting in a substantially higher P_{success} . This section therefore reports P_{path} as an illustrative metric, distinct from the PRISM-verified P_{success} , which was identical across runs.

The optimal path identified by PRISM therefore prioritises robustness and redundancy rather than single-path efficiency. In contrast, nearly all LLM-generated strategies lacked redundancy, relying instead on direct, single-pass routes. Even ChatGPT 5, which attempted to include contingency behaviour in Run 2, achieved only $P_{\text{success}} = 0.2383762$, lower than PRISM's verified result. This highlights the advantage of formal probabilistic reasoning in PRISM, where redundancy across multiple possible recovery paths improves the overall probability of meeting the goal compared to any single deterministic strategy.

Step	Movement	Updated resource locations	Allocation α
Init	—	$a=2, c=4, d=2; t_1 = a, t_2 = c$	$x_g = 0$
1	T2: $c \rightarrow g$ (4)	$a=2, d=2, g=4$	$x_g = 4$
2	T2: $g \rightarrow f$ (0)	$a=2, d=2, g=4$	$x_g = 4$
3	T2: $f \rightarrow e$ (0)	$a=2, d=2, g=4$	$x_g = 4$
4	T2: $e \rightarrow d$ (0)	$a=2, d=2, g=4$	$x_g = 4$
5	T1: $a \rightarrow d$ (1)	$a=1, d=3, g=4$	$x_g = 4$
6	T1: $d \rightarrow a$ (0)	$a=1, d=3, g=4$	$x_g = 4$
7	T1: $a \rightarrow d$ (1)	$a=0, d=4, g=4$	$x_g = 4$
8	T1: $d \rightarrow e$ (1)	$d=3, e=1, g=4$	$x_g = 4$
9	T1: $e \rightarrow d$ (0)	$d=3, e=1, g=4$	$x_g = 4$
10	T1: $d \rightarrow e$ (1)	$d=2, e=2, g=4$	$x_g = 4$
11	T1: $e \rightarrow d$ (0)	$d=2, e=2, g=4$	$x_g = 4$
12	T1: $d \rightarrow e$ (1)	$d=1, e=3, g=4$	$x_g = 4$
13	T2: $d \rightarrow e$ (1)	$d=0, e=4, g=4$	$x_g = 4$
14	T1: $e \rightarrow f$ (1)	$e=3, f=1, g=4$	$x_g = 4$
15	T1: $f \rightarrow e$ (0)	$e=3, f=1, g=4$	$x_g = 4$
16	T1: $e \rightarrow f$ (1)	$e=2, f=2, g=4$	$x_g = 4$
17	T1: $f \rightarrow e$ (0)	$e=2, f=2, g=4$	$x_g = 4$
18	T1: $e \rightarrow f$ (1)	$e=1, f=3, g=4$	$x_g = 4$
19	T2: $e \rightarrow f$ (1)	$e=0, f=4, g=4$	$x_g = 4$
20	T1: $f \rightarrow g$ (1)	$f=3, g=5$	$x_g = 4$
21	T1: $g \rightarrow f$ (0)	$f=3, g=5$	$x_g = 5$
22	T1: $f \rightarrow g$ (1)	$f=2, g=6$	$x_g = 5$
23	T1: $g \rightarrow f$ (0)	$f=2, g=6$	$x_g = 6$
24	T1: $f \rightarrow g$ (1)	$f=1, g=7$	$x_g = 6$
25	End	$f=1, g=7$	$x_g = 7$

Table 14: Step-by-step movements and resource allocations along the optimal path identified by PRISM.

7.3.2 The PRISM Path

The optimal path identified by PRISM, presented in Table 14, can be divided into several logical stages that reflect an optimisation purely for reliability. Although path lengths exist within the scenario, PRISM considers only the safety of each traversal and does not account for efficiency. The first stage involves Team 2 transporting all four resources from $c \rightarrow g$ in a single move along the red edge ($p = 0.5$). Importantly, failure at this stage results in an immediate scenario failure, since no remaining sequence of actions can deliver the required seven resources to g . This is a somewhat counterintuitive outcome, as one might expect the model to distribute the initial risk more evenly rather than concentrate it in a single high-stakes traversal.

Following the successful delivery of resources to g , Team 2 proceeds along $g \rightarrow f \rightarrow e \rightarrow d$, positioning itself to assist Team 1 or recover from potential

failures. Team 1 begins ferrying the two remaining resources between a and d , making multiple trips along the red $a \leftrightarrow d$ edge but carrying only one resource at a time. This gradual transfer reduces the loss per traversal if failure occurs, allowing subsequent actions to recover the target condition $x_g \geq 7$.

Once the resources at a and d have been consolidated, Teams 1 and 2 perform coordinated ferrying operations across the safer green $d \leftrightarrow e$ and $e \leftrightarrow f$ links, each moving one resource per trip. The final stage involves Team 1 completing several trips along the yellow $f \leftrightarrow g$ edge until seven resources have been successfully accumulated at g . Curiously, Team 2 never delivers any additional resources to g , even when only a single unit remains to satisfy K' .

This strategy demonstrates extensive contingency planning: after the initial transfer from c , any subsequent failure results in the loss of at most one resource, leaving recovery still possible. The verified strategy therefore minimises the probability of total mission failure by prioritising safe, incremental progress over path length or time, in contrast to the more direct, single-pass strategies produced by most LLMs.

7.3.3 Pipeline performance and reproducibility

All three PRISM pipeline runs completed successfully on a MacBook Pro equipped with an Apple M1 Max and 64 GB RAM. While the generated PRISM models differed in implementation details, their formal evaluations were identical, producing the same state and transition counts and the same verified properties. Table 15 summarises wall-clock time and token usage by stage.¹

Run	Verify time	Parser toks	Composer toks	Strategy toks
1	0:16:23.481845	2,771	46,195	10,039
2	0:09:11.101852	1,891	47,751	14,035
3	0:08:41.838508	2,292	46,684	16,428

Table 15: End-to-end performance of the PRISM pipeline by stage.

Notes on error recovery In Run 1 the initial PRISM model contained a syntax error. The first recovery attempt used *Auto-fix*, which produced a new model that still contained a syntax error. The pipeline then invoked its built-in error-handling sequence (*Auto-fix*, *Regenerate*, *Exit*), regenerating the model from scratch. The new model verified correctly.²

Interpretation The longer verification time in Run 1 is attributable to the error-recovery cycle rather than any difference in the verified results. The token counts shown correspond only to the initial model generation. The composer stage consistently dominated total token usage, reflecting that model synthesis

¹Token counts include both prompt and completion.

²All artefacts and logs are provided in Appendix A.

is the most verbose stage. Although the generated MDPs varied in syntax and structure, PRISM produced identical state and transition counts and the same P_{success} for all runs, confirming semantic equivalence across independently generated models.

7.3.4 Discussion

Compared with the large language model outputs, the PRISM pipeline achieved complete consistency across runs, producing the highest verified probability of success and the only strategy that incorporated full redundancy. Most LLM-generated plans produced plausible reasoning but lacked formal guarantees of optimality and often contained arithmetic or logical errors. PRISM’s probabilistic model checking evaluated the entire state space, identifying a strategy that balanced risk across multiple recovery paths and satisfied K' under the specified transition probabilities.

These results address the research objective of assessing whether LLMs can improve the usability of formal verification without compromising its rigour. While the LLMs studied could describe the scenario and generate syntactically valid PRISM models, they were less effective at evaluating the scenario itself. Their reasoning remained heuristic and occasionally incorrect, whereas PRISM’s verification provided a mathematically exhaustive analysis. This distinction illustrates that LLMs are best positioned as front-end tools that lower the expertise barrier for formal modelling rather than as substitutes for formal analysis.

The optimal strategy identified by PRISM is notable for how unintuitive it appears from a human perspective. It begins with a 50% chance of total failure, sending Team 2 across the riskiest red route before any redundancy is established. To a human planner, this seems reckless—effectively placing all resources at risk in a single move, or “putting all the eggs in one basket.” Yet this action is mathematically optimal when evaluated across the full state space, as it maximises the expected probability of ultimately reaching the goal state where $x_g \geq 7$. What seems reckless in isolation is optimal in aggregate, illustrating the difference between human intuition and formal probabilistic reasoning.

Overall, these findings suggest that combining LLMs and formal verification can meaningfully improve both accessibility without sacrificing reliability. LLMs can capture complex natural language scenarios and translate them into formal models, while tools such as PRISM ensure correctness through exhaustive evaluation. Together, they demonstrate a pathway toward making formal verification more usable and practical for real-world applications like disaster resourcing, where both ease of modelling and correctness are essential.

8 Conclusions and Future Work

This work investigated how large language models (LLMs) can be used to improve the usability of formal verification in the context of disaster resourcing.

The study proposed and implemented a pipeline that translates natural language descriptions of disaster scenarios into formal PRISM models, enabling automatic generation, verification, and strategy explanation. This approach sought to reduce the technical barrier traditionally associated with formal methods while maintaining the rigour and reliability that make them valuable for safety-critical applications.

The results show that while LLMs are effective at generating valid and executable PRISM models, they are unreliable when reasoning about probabilistic outcomes. Across all models tested, the LLM-generated strategies exhibited plausible but often incorrect logic, arithmetic errors, or an absence of redundancy. In contrast, PRISM’s verified strategy consistently achieved the highest probability of success and incorporated extensive contingency behaviour. These findings confirm that LLMs can assist in model creation but cannot yet replace formal verification as a means of ensuring correctness.

The combined system demonstrates that usability and rigour can coexist. By allowing a disaster manager to describe a scenario in natural language and automatically obtain a verified strategy, the pipeline makes formal verification more accessible to non-experts. This represents a meaningful step toward practical, human-centred formal verification.

Future work will focus on improving the reliability and interpretability of the LLM stages. This includes experimenting with alternative prompting strategies, exploring fine-tuned models for domain-specific language understanding, and incorporating intermediate validation steps. Extending the pipeline to handle continuous variables, temporal logic properties, and larger multi-agent systems would also increase its generality. Finally, evaluating performance across a broader range of disaster scenarios will provide a more comprehensive test of whether this proposed pipeline proves to be practicable.

References

- [1] Map viewer. [Online]. Available: <https://www.arcgis.com/apps/mapviewer/index.html?webmap=70d5e3bd84014665a965d37f5090507d>
- [2] K. Johnson and M. Kumari, “Probabilistic model checking of disaster resource distribution strategies,” in *2025 IEEE/ACM 13th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pp. 1–5, ISSN: 2575-5099. [Online]. Available: <https://ieeexplore.ieee.org/document/11024282>
- [3] F. Xu, J. Ma, N. Li, and J. C. Cheng, “Large language model applications in disaster management: An interdisciplinary review,” vol. 127, p. 105642. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S2212420925004662>
- [4] P. Shojaee, I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar, “The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity.” [Online]. Available: <http://arxiv.org/abs/2506.06941>
- [5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models.” [Online]. Available: <http://arxiv.org/abs/2201.11903>
- [6] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, “Sparks of artificial general intelligence: Early experiments with GPT-4.” [Online]. Available: <http://arxiv.org/abs/2303.12712>
- [7] J. Vendrow, E. Vendrow, S. Beery, and A. Madry, “Do large language model benchmarks test reliability?” [Online]. Available: <http://arxiv.org/abs/2502.03461>
- [8] M. Lott. Tracking AI. [Online]. Available: <https://www.trackingai.org>
- [9] C. Lee, D. Porfirio, X. J. Wang, K. Zhao, and B. Mutlu, “VeriPlan: Integrating formal verification and LLMs into end-user planning,” in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 1–19. [Online]. Available: <http://arxiv.org/abs/2502.17898>
- [10] Structured model outputs - OpenAI API. [Online]. Available: <https://platform.openai.com>
- [11] Civil defence centres map. [Online]. Available: <https://www.aucklandemergencymanagement.govt.nz/your-local/civil-defence-centres/>
- [12] K. Johnson, S. Madanian, and R. Sinha, “Graph-theoretic models of resource distribution for cyber-physical systems of disaster-affected regions,” in *2020 46th Euromicro Conference on Software Engineering*

- and *Advanced Applications (SEAA)*, pp. 521–528. [Online]. Available: <http://arxiv.org/abs/2112.12046>
- [13] K. Johnson, J. Cámara, R. Sinha, S. Madanian, and D. Parry, “Towards self-adaptive disaster management systems.”
 - [14] Strategies. [Online]. Available: <https://www.prismmodelchecker.org/manual/RunningPRISM/Strategies>
 - [15] C. Baier and J.-P. Katoen, *Principles of model checking*, ser. The MIT Press Ser. The MIT Press.
 - [16] E. W. Dijkstra, “A note on two problems in connexion with graphs,” vol. 1, no. 1, pp. 269–271. [Online]. Available: <https://doi.org/10.1007/BF01386390>
 - [17] Grok. [Online]. Available: <https://grok.com/>
 - [18] LMArena. [Online]. Available: <https://lmarena.ai>
 - [19] Google AI studio. [Online]. Available: <https://aistudio.google.com/>
 - [20] Claude. [Online]. Available: <https://claude.ai/new>
 - [21] Perplexity. [Online]. Available: <https://www.perplexity.ai/>
 - [22] DeepSeek. [Online]. Available: <https://chat.deepseek.com>

Appendices

A Code

All code and the data used for this study can be found at: <https://github.com/calebelson/NL-PRISM-Pipeline>

B Prompt

An example of the prompt given to define the main problem of this work is given below:

System Prompt

You are an expert in logistics and supply chain management, specializing in optimizing resource allocation and routing under constraints. You have a deep understanding of graph theory, probability, and optimization techniques.

Two teams: T1 at a, and T2 at c, each can carry 4 resources at a time. Point c has 4 resources, a and d each have 2. Point g wants 7 resources.

Routes: a-b red distance 9, a-d red distance 5, b-c red distance 4, b-d yellow distance 12, c-g red distance 6, c-f red distance 12, d-e green distance 2, e-f green distance 4, f-g yellow distance 10. Undirected.

Safety probabilities are green = 0.99, yellow = 0.85, red = 0.5. Objective is to maximise probability of reaching the goal.

If a team fails, the team is no longer available and any resources it is carrying are lost. If it is no longer possible for the goal to be reached, the plan fails.

Task:

1. Determine whether the available supplies can satisfy the demands at g.
2. If feasible, produce a routing and allocation plan for Team 1 and Team 2 that maximizes the possibility of the desired resources reaching their goal.
3. You should optimize exclusively for reliability.
4. If no feasible plan exists, explain why.
5. Give the probability of success for your plan.
6. Provide a step-by-step description of your plan.