Evaluating Large Language Models for a Situation–Planning Agent in Rail Logistics

Huy (David) Doan
Student Number 240428200
MSc. Artificial Intelligence, QMUL
ec24154@qmul.ac.uk
Project Supervisor: Professor Matthew Purver
m.purver@qmul.ac.uk

The TRAINS domain began as a mixed-initiative planning testbed in the 1990s. It paired a conversational assistant with a rail-logistics world to study collaborative plans (*TRAINS Spoken Dialog Corpus (LDC95S25)* 1995; Ferguson et al., 1996; Allen et al., 1995). Tasks require clear grounding, resource control, and time management. They also require simple manufacturing rules. Earlier work struggled with speech, language, and real-time planning.

We revisit this setting with a hybrid planner. The system uses an LLM (OpenAI `gpt-4o` via DSPy). It outputs plans in a constrained JSON format. A patch layer enforces routing, capacity, and conversion rules. A greedy fallback proposes a plan if the patched plan still fails.

We evaluate 18 tasks with deadlines (one task is excluded as unsolvable). Success rises from 33.3% with the LLM alone to 83.3% with patches and to 88.9% with the fallback. The criterion is strict: exact deliveries, no violations, and on-time arrival. The main contribution is architectural. We pair *declarative intent* from a language model with *executable guarantees* from code. The pattern gives dependable plans and a simple path to reproduction (Yao et al., 2022; Yao et al., 2023; Khattab et al., 2023; Ahn et al., 2022; Liang et al., 2022; Lu et al., 2020).

***Index Terms*—LLM planning, hybrid reasoning, rail logistics, error patching, constrained decoding**

I. Introduction

Large language models now act as planners. They generate action sequences or small programs. Naïve prompting often fails in practice. It may skip legal hops. It may ignore capacity. It may miss a key step. ReAct and Tree-of-Thoughts improve reliability by adding structure (Yao et al., 2022; Yao et al., 2023). DSPy improves pipelines by making them typed and optimisable (Khattab et al., 2023).

We study a compact rail-logistics domain. Engines move on a fixed network. Inventory is tied to locations. Orange juice is produced at one site only. Each task has a deadline (*TRAINS Dialogue Corpus* n.d.; *CISD: TRAINS Project Overview* n.d.). Our system has three parts. The LLM emits JSON plans. The patch layer repairs common errors. The fallback builds a simple feasible plan when needed. We assess 19 tasks in total and evaluate 18 of them. A run is successful only when all deliveries are exact, all rules hold, and all deadlines are met. The hybrid approach is light, reproducible, and effective.

II. Background and Related Work

A. LLM agents and structured prompting

ReAct mixes thoughts with actions. It reduces hallucinations and enables correction (Yao et al., 2022). Tree-of-Thoughts explores several reasoning paths. It adds deliberate lookahead and simple backtracking (Yao et al., 2023). We follow the spirit of these methods. We also constrain output to a typed schema and validate it after generation.

B. Programmatic LM pipelines

DSPy compiles module graphs into LM pipelines. It tunes prompts and parameters. It also improves reliability over ad-hoc chains (Khattab et al., 2023). We use DSPy signatures to force JSON only, expose constraints, and compose non-LM code with the planner.

C. Classical planning and search

Simple graph algorithms still matter for reliability. We use BFS/Dijkstra for routing and choose tasks by earliest deadline. This fallback restores feasibility when the plan is under-specified (Dijkstra, 1959; Cormen et al., 2009). It complements the LLM.

D. TRAINS-style logistics domains

The TRAINS project created a strong testbed for mixed-initiative planning. It supports reasoning about time, actions, and resources (*TRAINS Dialogue Corpus* n.d.; *CISD: TRAINS Project Overview* n.d.). We adopt a simplified variant. The topology is fixed. Inventories and facilities sit at known sites. We can therefore check success automatically.

III. Problem Setting and Benchmark

A. Domain

We study a constrained rail–logistics domain on a compact network of five locations: *Avon*, *Dansville*, *Corning*, *Bath* and *Elmira*. Engines haul boxcars and tankers along a fixed topology with legal adjacencies only (no skipping of intermediate nodes). Commodities and facilities are location–specific:

- **Bananas** load *only* at *Avon*.

- **Oranges** load *only* at *Corning*.
- **Orange juice** is produced *only* at *Elmira* by converting loaded oranges in the presence of empty tankers.

An engine may haul at most **three loaded cars** at any time (boxcars carrying cargo and tankers containing juice). Empty rolling stock does not count towards this limit. Each problem instance specifies a set of deliveries (quantities per commodity and destination) and hard **deadlines**.

B. Action Schema and Legality

Plans are expressed as a typed JSON sequence of actions from `START`, `ATTACH`, `TRAVEL`, `LOAD`, `UNLOAD`, `CONVERT`, `DETACH`. Legality requires: (i) **routing** along a single legal edge per `TRAVEL`; (ii) **preconditions** (engines co-located with resources before `ATTACH`/`LOAD`/`CONVERT`); (iii) **capacity** $\leq$ three *loaded* cars; (iv) **manufacturing** at Elmira only (converting oranges $\rightarrow$ juice and filling attached tankers); and (v) **deadlines** met.

C. Benchmark

We evaluate **19** TRAINS-style tasks; one (2-C) is excluded as unsolvable; **18** are assessed. Tasks are grouped by nominal difficulty:

- **Simple (5):** W, 1-A, 1-B, 1-C, 1-E
- **Medium (6):** 2-A, 2-B, 2-D, 2-E, 2-F, 2-G
- **Complex (7):** 1-D, 3-A, 3-B, 3-C, 3-D, 3-E, 3-F
- **Excluded:** 2-C

Every evaluated problem includes an explicit deadline.

D. Success Criterion

A run *counts as solved* only if the generated plan (possibly post-processed by our repair and fallback components, §IV) satisfies all deliveries exactly, respects capacity and manufacturing constraints throughout execution, uses only legal edges, and arrives before all deadlines.

IV. Methods

A. System Overview

The pipeline has three stages. The planner emits a structured action sequence. The patch layer repairs common failure modes while preserving intent. The fallback synthesises a feasible plan when validation still fails. A single checker validates each candidate. Figure 1 shows the flow.

B. LLM Planner (via DSPy)

We call OpenAI `gpt-4o` through DSPy (Khattab et al., 2023). Decoding is deterministic. We set `temperature=0` and `max_tokens=2000`. The signature demands JSON output. It also embeds reminders on legal hops, loading sites, conversion rules, capacity, and multi-engine use when needed.

C. Validation

We execute each plan symbolically. We then check edges, preconditions, capacity, conversion at Elmira, and exact on-time delivery. A plan passes only if all checks pass.

D. Patching Layer

Seven repairs apply until no further change is needed:

- **P1:** fix invalid `START` and missing `TRAVEL` steps.
- **P2:** expand illegal multi-hop moves into legal per-edge `TRAVEL`.
- **P3:** split over-capacity segments across engines or trips.
- **P4:** align `CONVERT` quantities with oranges and tankers.
- **P5:** separate bundled deliveries and insert `DETACH` when needed.
- **P6:** normalise action order and add any missing terminal steps.
- **P7:** remove actions after the goal is achieved.

E. Algorithmic Fallback

When a patched plan fails the checker, we build a new plan. We use BFS/Dijkstra for routes and earliest-deadline-first for tasks (Dijkstra, 1959; Cormen et al., 2009). We also split by capacity. The approach is simple by design.

F. Dataset and System Map

The benchmark includes 18 evaluated tasks and a fixed rail network. A schematic map of nodes and legal edges is provided in the appendix.

G. Experimental Protocol

We test four configurations. Baseline uses the LLM only. Strategic adds hints but no patches. Patches adds deterministic repair. Full combines patches and fallback. A run counts as solved only if §III holds. The code uses Python 3.12.7 with `dspy-ai`, `python-dotenv`, and `openai`. The repository is public. Users must provide their own API keys.

V. Results

A. Headline Performance

Overall success rates appear in Table I. The baseline solves 6 of 18 tasks (33.3%). Adding patches lifts accuracy to 83.3% (+50.0 percentage points). The full system reaches 88.9% (+55.6 percentage points). These gains require no training and no sampling. We fix temperature at zero. Structure, not randomness, drives reliability (Khattab et al., 2023; Wang et al., 2022).

B. Category-level Effects

With the given difficulty split, we see three patterns. First, the baseline already solves all five simple tasks. Patches act as safety checks. Second, patches unlock four medium tasks on their own, and synergy with strategy is needed for 2-A, 2-D, and 2-E. Third, patches solve 1-D, 3-A, and 3-B among complex tasks. Case 3-E needs strategy plus patches. Three tasks remain unsolved (3-C, 3-D, 3-F).

C. Ablation: Where the Gains Come From

Ablation identifies P1, P5, and P6 as the main drivers. Removing any one pushes results back toward the baseline. P1 fixes starts and missing hops. P5 adds multi-stop structure. P6 enforces completion and order. Their high trigger rate reflects global checks, not constant edits (Lu et al., 2020; Lu et al., 2021).

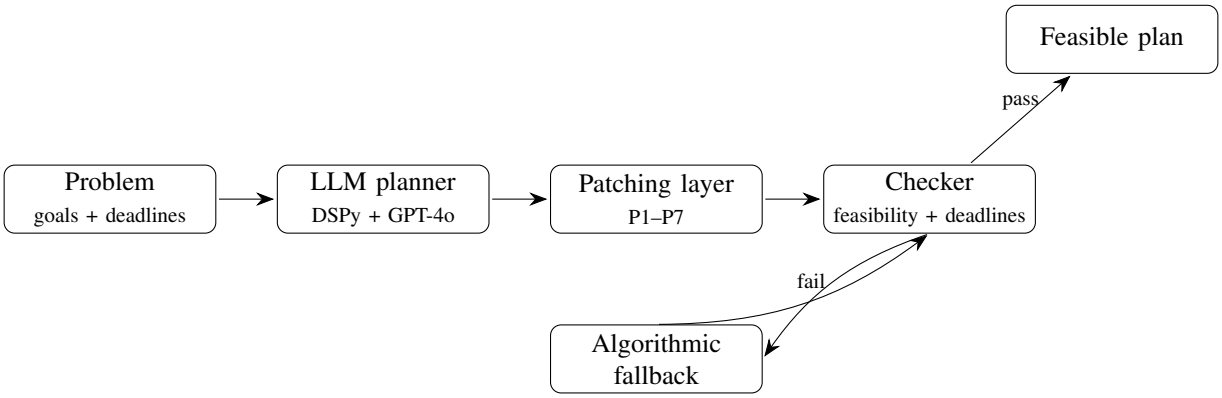

Fig. 1. System architecture. The planner proposes a plan; deterministic patches repair common errors; the checker enforces feasibility (routing, capacity, conversion, deadlines). On failure, a greedy fallback generates a new candidate, which is rechecked.

D. Efficiency and Error Reduction

The system removes three error families. It replaces illegal edges with per-edge `TRAVEL`. It splits loads to respect the three-car capacity. It aligns `CONVERT` with quantities and site. The enhanced mode uses about 601 tokens per task. The estimated cost is about $0.0047 per task. Patching and checking add little overhead (Shinn et al., 2023).

E. Case Studies

We summarise three patterns. 2-E succeeds with planned 3+2 splitting (synergy). 1-D is fixed by patches alone. 3-C remains unsolved under a single-engine rule. Full JSON snippets appear in Appendix B.

VI. Discussion

A. Significance of the research findings

The study shows that reliable planning is possible with simple tools. We pair a structured LLM with a small set of repairs and a greedy fallback. Accuracy rises from 33.3% to 88.9% (**??**). Three repairs (P1/P5/P6) deliver most gains. They convert many medium and complex tasks from failure to success (Tables II and III). The pipeline is cheap to run and easy to audit.

B. Comparison with existing literature

Prior work improves reliability by adding reasoning or control. ReAct mixes thoughts and actions (Yao et al., 2022). Tree-of-Thoughts explores several branches (Yao et al., 2023). DSPy builds typed pipelines (Khattab et al., 2023). SayCan and Code-as-Policies ground actions in skills or programs (Ahn et al., 2022; Liang et al., 2022). NeuroLogic constrains decoding during generation (Lu et al., 2020; Lu et al., 2021). We differ by enforcing constraints after generation with light code. We also keep decoding deterministic. The result is strong reliability at low cost.

C. Potential explanations or hypotheses

H1. A few error types dominate. P1, P5, and P6 target them directly. **H2.** Strategy and patches interact. Hints lower structural error. Patches then finish the plan. **H3.** Determinism helps enforcement. Fixed decoding makes checking predictable and effective (Wang et al., 2022).

D. Limitations and future research

The benchmark is small and stylised. It contains 18 evaluated tasks in a fixed world (Ferguson et al., 1996; *TRAINS Spoken Dialog Corpus (LDC95S25)* 1995). Three cases remain unsolved (3-C, 3-D, 3-F). Local repair is not enough for global scheduling or conversion conflicts. Future work will add light lookahead, stronger decoding constraints, and learning from failure. We also plan to test larger graphs, variable times, and delays.

E. Consistency with the original hypotheses

Results support all three hypotheses. H1 holds: failures cluster into a few local patterns, and P1/P5/P6 address them. H2 holds within scope: strategy alone adds no solves, but strategy plus patches unlocks 2-A, 2-D, 2-E, and 3-E. H3 holds: zero-temperature decoding reduces variance and helps post-hoc checks (Wang et al., 2022). We also see two anomalies. P3 rarely triggers because earlier steps and signature reminders prevent capacity errors. Case 2-G regresses under patches-only; the full system recovers it.

F. Comparison with recent literature (past five years)

The broad lesson agrees with recent work. Out-of-the-box LLMs struggle at planning, but structure helps (Yao et al., 2022; Yao et al., 2023; Khattab et al., 2023). Our approach chooses post-generation guarantees rather than token-level control. It fits closed, rule-rich tasks well. It also suits tight token budgets.

G. Theoretical contributions and innovations

We offer four ideas. First, a pattern: *LLM for intent; code for guarantees*. Second, a minimal patch basis: P1/P5/P6. Third, synergy without sampling: hints plus repairs. Fourth, a

template: checker–patcher–fallback. Holding temperature fixed lets us measure cost–reliability trade-offs clearly.

VII. Conclusion

We presented a hybrid planner for rail logistics. The system combines a structured LLM, deterministic repairs, and a greedy fallback. On 18 evaluated tasks with deadlines, success rises from 33.3% to 83.3%, and to 88.9% with the fallback. Ablations identify P1/P5/P6 as the key repairs. Strategy and patches unlock the hardest cases.

Future work will add light global search, decoding constraints, and learning from failure. We also plan to scale the benchmark. The lesson is simple. Combine *declarative intent* from an LLM with *executable guarantees* from code to get dependable plans.

Appendix A
Leave-One-Out and Necessity (Full Tables)

This appendix contains the full comparison and ablation tables: configuration performance (Table I), patch necessity (Table II), and leave-one-out removal effects (Table III).

TABLE I
Configuration performance (exact lists included).

Configuration	Solved	Rate	Problems
Baseline	6/18	33.3%	W, 1-A, 1-B, 1-C, 1-E, 2-G
Strategic only	6/18	33.3%	W, 1-A, 1-B, 1-C, 1-E, 2-G
Patches only	10/18	55.6%	W, 1-A, 1-B, 1-C, 1-D, 1-E, 2-B, 2-F, 3-A, 3-B
Full system	15/18	83.3%	W, 1-A, 1-B, 1-C, 1-D, 1-E, 2-A, 2-B, 2-D, 2-E, 2-F, 2-G, 3-A, 3-B, 3-E

TABLE II
Patch necessity on patch-dependent problems (corrected).

Patch	Triggers	Needed	Over-trig.	Role
P1	13/18	**5/5**	23%	Critical (engine placement)
P5	18/18	**5/5**	33%	Critical (multi-destination)
P6	18/18	**5/5**	33%	Critical (completion/order)
P4	2/18	2/5	0%	Specific (convert qty)
P7	6/18	0/5	100%	Cleanup only
P2	1/18	0/5	0%	Rare edge case
P3	0/18	0/5	–	Not used

TABLE III
Leave-one-out (removal) effect.

Configuration	Solved	Rate	Impact
Baseline (no patches)	6/18	33.3%	–
All patches	10/18	55.6%	+4
Without **P1**	6/18	33.3%	Loses all 5 patch gains
Without **P5**	6–7/18	∼36%	Loses 3–4 problems
Without **P6**	6–7/18	∼36%	Loses 3–4 problems
Without **P4**	8/18	44.4%	Loses 1-D, 2-B
Without P2,P3,P7	10/18	55.6%	No change

Appendix B
Case Study JSON (Wrapped, Simplified)

A. 2-E (Synergy) — before/after (illustrative).

```
{"steps":[
  {"act":"START","args":{"engine":"E2","at":"Elmira"}},
  {"act":"ATTACH","args":{"engine":"E2","car_type":"
    ↪ boxcar","qty":2}},
  {"act":"LOAD","args":{"engine":"E2","at":"Corning",
                        "cargo":"oranges","cars":2}},
  {"act":"TRAVEL","args":{"engine":"E2","from":"Corning
    ↪ ","to":"Bath"}}
]}
```

```
{"steps":[
  {"act":"START","args":{"engine":"E2","at":"Elmira"}},
  {"act":"START","args":{"engine":"E3","at":"Elmira"}},
  {"act":"ATTACH","args":{"engine":"E2","car_type":"
    ↪ boxcar","qty":3}},
  {"act":"ATTACH","args":{"engine":"E3","car_type":"
    ↪ boxcar","qty":2}},
  {"act":"LOAD","args":{"engine":"E2","at":"Corning","
    ↪ cargo":"oranges","cars":3}},
  {"act":"LOAD","args":{"engine":"E3","at":"Corning","
    ↪ cargo":"oranges","cars":2}},
  {"act":"TRAVEL","args":{"engine":"E2","from":"Corning
    ↪ ","to":"Bath"}},
  {"act":"TRAVEL","args":{"engine":"E3","from":"Corning
    ↪ ","to":"Bath"}}
]}
```

B. 1-D (Patches-only) — baseline vs. patched.

```
{"steps":[
  {"act":"START","args":{"engine":"E1","at":"Bath"}},
  {"act":"ATTACH","args":{"engine":"E1","car_type":"
    ↪ boxcar","qty":3}},
  {"act":"LOAD","args":{"engine":"E1","at":"Corning",
                        "cargo":"oranges","cars":3}}
]}
```

```
{"steps":[
  {"act":"START","args":{"engine":"E1","at":"Elmira"}},
  {"act":"ATTACH","args":{"engine":"E1","car_type":"
    ↪ boxcar","qty":3}},
  {"act":"LOAD","args":{"engine":"E1","at":"Corning","
    ↪ cargo":"oranges","cars":3}},
  {"act":"ATTACH","args":{"engine":"E1","car_type":"
    ↪ tanker","qty":3}},
  {"act":"CONVERT","args":{"engine":"E1","at":"Elmira","
    ↪ qty":3}},
  {"act":"TRAVEL","args":{"engine":"E1","from":"Elmira","
    ↪ to":"Bath"}}
]}
```

C. 3-C (Single engine) — attempt still fails.

```
{"steps":[
  {"act":"START","args":{"engine":"E2","at":"Elmira"}},
  {"act":"ATTACH","args":{"engine":"E2","car_type":"
    ↪ boxcar","qty":3}},
  {"act":"LOAD","args":{"engine":"E2","at":"Avon","cargo
    ↪ ":"bananas","cars":3}},
  {"act":"TRAVEL","args":{"engine":"E2","from":"Avon","to
    ↪ ":"Bath"}},
  {"act":"TRAVEL","args":{"engine":"E2","from":"Bath","to
    ↪ ":"Corning"}},
  {"act":"LOAD","args":{"engine":"E2","at":"Corning","
    ↪ cargo":"oranges","cars":2}},
  {"act":"CONVERT","args":{"engine":"E2","at":"Elmira","
    ↪ qty":2}}
]}
```

References

Ahn, Michael et al. (2022). "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv:2204.01691*.

Allen, James F. et al. (1995). "TRAINS as an Embodied Natural Language Dialog System". In: *AAAI Fall Symposium on Embodied Language and Action*. Accessed 2025-08-19. URL: https://cdn.aaai.org/Symposia/Fall/1995/FS-95-05/FS95-05-001.pdf.

CISD: TRAINS Project Overview (n.d.). https://www.cs.rochester.edu/research/cisd/projects/trains/. Accessed 2025-08-19.

Cormen, Thomas H. et al. (2009). *Introduction to Algorithms*. 3rd ed. MIT Press.

Dijkstra, Edsger W. (1959). "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1, pp. 269–271.

Ferguson, George, Allen, James F., and Miller, Brad (1996). "TRAINS-95: Towards a Mixed-Initiative Planning Assistant". In: *Proceedings of the Third International Conference on AI Planning Systems (AIPS-96)*. Accessed 2025-08-19. URL: https://www.cs.rochester.edu/research/cisd/pubs/1996/ferguson-allen-miller-aips96.pdf.

Khattab, Omar et al. (2023). "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines". In: *arXiv:2310.03714*.

Liang, Jacky et al. (2022). "Code as Policies: Language Model Programs for Embodied Control". In: *arXiv:2209.07753*.

Lu, Xuebo et al. (2020). "NeuroLogic Decoding: (Un)supervised Neural Text Generation with Predicate Logic Constraints". In: *arXiv:2010.12884*.

– (2021). "Constrained Text Generation with Lookahead Heuristics". In: *arXiv:2112.08726*.

Shinn, Noah et al. (2023). "Reflexion: Language Agents with Verbal Reinforcement Learning". In: *arXiv:2303.11366*.

TRAINS Dialogue Corpus (n.d.). https://www.cs.rochester.edu/research/speech/trains.html. Accessed 2025-08-19.

TRAINS Spoken Dialog Corpus (LDC95S25) (1995). Linguistic Data Consortium. Accessed 2025-08-19. URL: https://catalog.ldc.upenn.edu/LDC95S25.

Wang, Xuezhi et al. (2022). "Self-Consistency Improves Chain of Thought Reasoning in Language Models". In: *arXiv:2203.11171*.

Yao, Shunyu et al. (2022). "ReAct: Synergizing Reasoning and Acting in Language Models". In: *arXiv:2210.03629*.

– (2023). "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *arXiv:2305.10601*.