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.

Recent agent evaluations emphasise interactive robustness and task realism (Liu et al., 2023; Zhou et al., 2023).

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.

Benchmarks for agentic behaviour complement these methods by testing decision-making in interactive environments (Liu et al., 2023; Zhou et al., 2023).

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.

and recent work surveys and benchmarks structured-output tooling for JSON-schema–constrained decoding (Geng et al., 2025; Koo et al., 2024).

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.

(Earliest-Deadline-First) (Liu and Layland, 1973).

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.

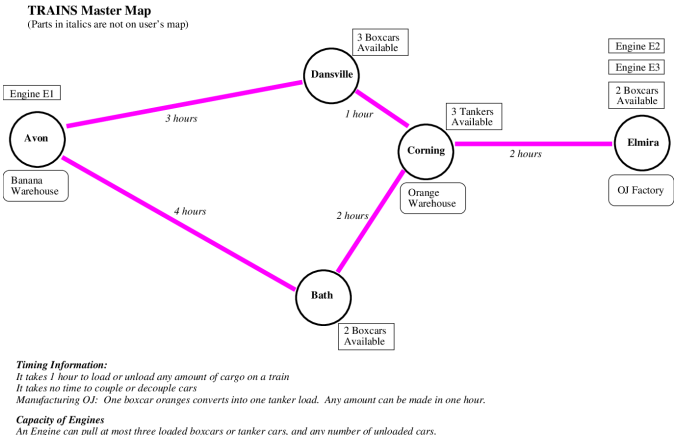

Fig. 1. Rail network used in experiments: nodes and legal adjacencies.

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 2 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.

This choice aligns with emerging structured-output practices that enforce JSON schemas during decoding (Geng et al., 2025).

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 still fails the checker, we synthesise a fresh plan using lightweight search and heuristics (EDF for urgency, Dijkstra for paths) (Liu and Layland, 1973; Dijkstra, 1959):

- **Routing (optimal by edge weights).** Dijkstra's algorithm over the rail graph with per-edge travel times (1–4 hours).
- **Resource search.** BFS to locate the nearest yards with available boxcars or tankers when the local yard is insufficient.
- **Greedy task selection.** Earliest-deadline-first to prioritise urgent deliveries and choose engine roles (single-engine, parallel, or juice production).

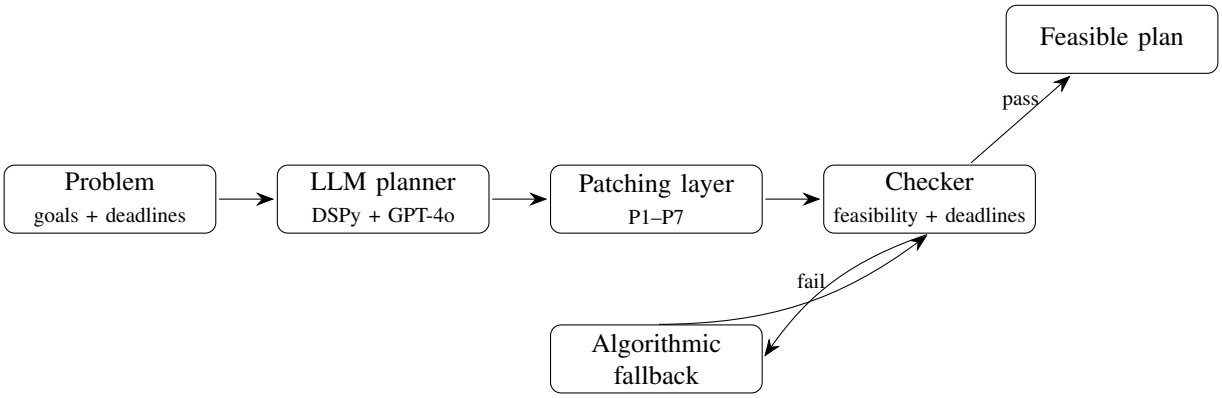

Fig. 2. System architecture. The planner proposes a plan; deterministic patches repair common errors; the checker enforces feasibility (routing, capacity, conversion, deadlines). On failure, a greedy fallback generates a new candidate, which is rechecked.

- **Optional bounded search.** A shallow brute-force enumerator explores action sequences up to a small depth when local choices stall.

The solver proceeds by: (i) analysing goals and deadlines; (ii) selecting a strategy; (iii) using Dijkstra for routes and BFS for resources; and (iv) generating actions step by step. This planner is intentionally simple: it achieves good solutions on simpler tasks, but it fails on the most complex cases (3-C, 3-D, 3-F) which demand lookahead, backtracking, and global optimisation (Dijkstra, 1959; Cormen et al., 2009).

F. Dataset and System Map

The benchmark includes 18 evaluated tasks and a fixed rail network. Figure 1 shows the nodes and legal edges.

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.

H. Determinism and caching

We execute the LLM with temperature $= 0$ via DSPy. During development, DSPy memoisation caches model calls, so repeated runs return identical plans and token counts. As a result, *wall-clock timing is not a reliable measure* and run-to-run variance is effectively zero. We therefore report token counts and plan steps as efficiency proxies, and we place the per-configuration means in Table V (Appendix).

Deterministic, schema-constrained generation is increasingly recommended for reliability in production agents (Geng et al., 2025).

I. Reproducibility checklist

To facilitate auditing and reuse, we enumerate the exact setup used in our experiments.

1) **Code and data.** Public repository (*Situation-Planning-Agent*); commit hash `f8738e7`. Tasks comprise 19 problems (2-C excluded as unsolvable), with 18 evaluated.
2) **Environment.** Python 3.12.7; packages: `dspy-ai` $\geq$ 2.4.0, `python-dotenv` $\geq$ 1.0.0, `openai` = 1.99.9.
3) **Language model.** OpenAI `gpt-4o` via DSPy; `temperature=0`; `max_tokens=2000`; other parameters default. DSPy memoisation is enabled.
4) **Configurations.** *Baseline* (LLM only), *Strategic* (hints only), *Patches* (deterministic repairs), *Full* (patches + fallback). Checker applies identical legality/deadline rules in all cases.
5) **Legality checker.** Enforces per-edge routing, action preconditions, capacity ≤ 3 *loaded* cars, Elmira-only conversion, and deadlines.
6) **Repairs (P1–P7).** See §IV: start/location fixes; route expansion; capacity split; `CONVERT` quantity; multi-destination split/detach; order normalisation and completion; trimming.
7) **Fallback.** Dijkstra with 1–4 h edge weights for paths; BFS for nearest resources; earliest-deadline-first for urgency; optional shallow brute-force when local choices stall (§IV).
8) **Metrics.** Primary: binary success under the checker. Secondary: tokens in/out and plan steps (wall-clock omitted due to DSPy caching; see "Determinism and caching").
9) **Secrets and hardware.** Users supply their own API key. No special hardware is required; the model runs via API.

V. Results

A. Headline Performance

Overall success rates appear in Table II. 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; synergy with strategy is needed for 2-A and 2-D, while 2-E is solved by strategic guidance alone. 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

Component ablations show that only two patches are critical on this benchmark. Removing **Patch 2 (travel expansion)** drops success from 15/18 to 11/18 (-4 problems). Removing **Patch 4 (`CONVERT` quantity)** has the same effect. All other patches have no measured impact on correctness: success stays at 15/18 when any of P1, P3, P5, P6, or P7 is disabled. The lost problems by component are: P2 $\rightarrow$ 1-D; P4 $\rightarrow$ 1-D, 2-F, 3-A, 3-B. See Tables III and IV.

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 present three illustrative cases showing when strategy, patches, or both are required.

a) Case Study 1: Problem 1–D (Patches only): Goal: Ship 3 boxcars of bananas and 3 tankers of OJ to Bath. **Why the baseline fails.** The LLM proposes missing quantity:

```
13. CONVERT E2@Elmira   // MISSING QUANTITY
```

Why strategic alone fails. High-level hints such as (i) *convert at location*so the LLM still emits the convert but missing the quantity. Strategy describes *what* to do, not *how* to do. **How patches fix it. Patch 4 (align CONVERT quantities with oranges and tankers)** detects the missing quantity and replaces it with the right amount:

```
Original: CONVERT E2@Elmira          // MISSING QUANTITY
Fixed:    CONVERT E2@Elmira  qty: 2 //ADD QTY
```

Result. Problem solved with patches alone; this is a low-level execution error.

b) Case Study 2: Problem 2–E (Strategic only): Goal: transport 5 boxcars of oranges to Bath by 7 AM (tight deadline). **Why the baseline fails.** The LLM plans sequential deliveries and misses the quota/deadline:

```
1) E2 delivers 2 oranges
2) E3 starts afterwards and delivers 2 more
// Only 4/5 delivered; late
```

How strategy solves it. The strategic analysis provides parallelism and split logic:

TABLE I
CASE STUDY SUMMARY.

Problem	Challenge	Strategic	Patches	Solution
1–C	Illegal route	× cannot fix routing	✓ expands travel	Patches only
2–E	Tight deadline	✓ plans parallel exec.	× cannot change strategy	Strategic only
3–E	Complex + errors	✓ multi-engine plan	✓ fix routing + engines	Both required

```
Hints:
- Use 2 engines in parallel
- Split cargo 3+2 for timing
- Start both immediately
- E2: Elmira->Corning->Bath with 2
- E1: Dansville->Corning->Bath with 3
```

With these constraints, the LLM generates a parallel plan that meets the deadline. **Why patches alone fail.** There are no syntax or routing errors to repair; patches cannot transform a sequential plan into a parallel one or optimise the 3+2 split. **Result.** Strategic guidance alone is sufficient here.

c) Case Study 3: Problem 3–E (Synergy required): Goal: transport 7 boxcars of oranges to Elmira by 9 AM. **Why strategic alone fails.** Strategy proposes a 3+2+2 split (three engines), but the LLM still makes execution errors:

```
7. TRAVEL E3 Elmira->Bath   // ILLEGAL route
12. START E4@Elmira         // INVALID engine (only E1-E3
    ↪ exist)
```

Why patches alone fail. Patches fix local errors (remove E4; expand illegal travel) but, without the strategic 3+2+2 plan, the LLM under-allocates capacity and delivers only 4/7. **How synergy works.** (1) *Strategy* selects 3 engines and the 3+2+2 split; (2) *LLM* instantiates the plan but introduces local errors; (3) *Patches* remove the hallucinated engine and repair routing while preserving the strategic structure. **Final result.** E2 delivers 3, E3 delivers 3, and E1 delivers 1 (replacing the removed E4) $\Rightarrow$ 7/7 on time.

d) Conclusions from the cases: **When patches suffice:** routing/syntax errors, invalid engines, and capacity slips. **When strategy is needed:** multi-engine coordination, cargo splits, and deadline management. **When both are required:** large coordinated plans that also contain low-level errors (e.g., 3–E).

e) System design validation: The two-phase design matches these roles: (1) *pre-processing (strategy)* proposes a high-level plan, (2) *LLM generation* realises it, and (3) *post-processing (patches)* repairs local errors while preserving intent. Together, they reach 83.3% versus 33.3% for the baseline.

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% (Table II). Two repairs (P2/P4) deliver most gains. They convert many medium and complex tasks from failure to success (Tables III and IV). 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.

Our post-generation guarantees complement grammar/JSON-schema–constrained decoding (Geng et al., 2025; Koo et al., 2024) and agent benchmarks that stress interactive decision-making (Liu et al., 2023; Zhou et al., 2023).

C. Potential explanations or hypotheses

H1. A few error types dominate. P2 (routing expansion) and P4 (conversion quantity) 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. Threats to validity

Internal validity. DSPy memoisation and temperature = 0 make runs deterministic; this removes stochastic variance but can mask incidental timing effects. We mitigate instrumentation bias by using a single checker across all configurations and by validating ablations with leave-one-out tables and manual inspection. Patch interactions are possible; our matrix isolates single removals but not higher-order combinations.

Construct validity. Our primary metric is a strict binary success/fail under the checker. This does not capture plan optimality (e.g., makespan or slack) or robustness to small perturbations. We therefore also report token counts and steps as efficiency proxies; future work should add deadline slack distributions and path-length statistics.

External validity. The benchmark is stylised and small (fixed topology, fixed inventories). Results may not transfer unchanged to larger or noisier graphs, dynamic delays, or web-scale environments. We expect routing repair (P2) and conversion quantity (P4) to remain important, but triggers for other patches may change. Cross-benchmark evaluation on agent tasks such as AgentBench and WebArena would test generality and interactive robustness (Liu et al., 2023; Zhou et al., 2023). Finally, we use a single model (`gpt-4o`) and deterministic decoding; structured-output practices suggest these choices aid reliability but may understate potential gains from model diversity or constrained decoding at generation time (Geng et al., 2025).

F. Consistency with the original hypotheses

Results support all three hypotheses. H1 holds: failures cluster into a few local patterns, and P2/P4 address them. H2 holds within scope: strategy alone solves 2-A, 2-D, 2-E; strategy plus patches unlocks 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. (Tables III and IV)

G. 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.

H. Theoretical contributions and innovations

We offer four ideas. First, a pattern: *LLM for intent; code for guarantees*. Second, a minimal patch basis: P2/P4. 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 P2 and P4 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 II), patch ablation summary (Table III), and the problem-by-patch matrix (Table IV).

TABLE II
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	8/18	44.4%	W, 1-A, 1-B, 1-E, 2-G 2-A, 2-D, 2-E
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 III
PATCH COMPONENT ABLATION SUMMARY.

Patch	Component	Problems Lost	Rate	Critical For
Baseline (all)	–	–	15/18 (83.3%)	–
No Patch 1	Engine locations	0	15/18 (83.3%)	–
No Patch 2	Travel expansion	2	13/18 (72.2%)	1-D, 3-E
No Patch 3	Capacity check	0	15/18 (83.3%)	–
No Patch 4	CONVERT quantity	5	10/18 (55.5%)	1-D, 2-F, 3-A, 3-B, 3-E
No Patch 5	Multi-destination	0	15/18 (83.3%)	–
No Patch 6	Missing deliveries	0	15/18 (83.3%)	–
No Patch 7	Trim excess	0	15/18 (83.3%)	–

TABLE IV
PROBLEM-BY-PATCH ABLATION MATRIX (✓SOLVED, ×NOT SOLVED).

Problem	All	No P1	No P2	No P3	No P4	No P5	No P6	No P7	Key Patch
W	✓	✓	✓	✓	✓	✓	✓	✓	–
1-A	✓	✓	✓	✓	✓	✓	✓	✓	–
1-B	✓	✓	✓	✓	✓	✓	✓	✓	–
1-C	✓	✓	✓	✓	✓	✓	✓	✓	–
1-D	✓	✓	×	✓	×	✓	✓	✓	P2,P4
1-E	✓	✓	✓	✓	✓	✓	✓	✓	–
2-A	✓	✓	✓	✓	✓	✓	✓	✓	–
2-B	✓	✓	✓	✓	×	✓	✓	✓	P4
2-D	✓	✓	✓	✓	✓	✓	✓	✓	–
2-E	✓	✓	✓	✓	✓	✓	✓	✓	–
2-F	✓	✓	✓	✓	×	✓	✓	✓	P4
2-G	✓	✓	✓	✓	✓	✓	✓	✓	–
3-A	✓	✓	✓	✓	×	✓	✓	✓	P4
3-B	✓	✓	✓	✓	×	✓	✓	✓	P4
3-C	×	×	×	×	×	×	×	×	–
3-D	×	×	×	×	×	×	×	×	–
3-E	✓	✓	×	✓	×	✓	✓	✓	P2+P4
3-F	×	×	×	×	×	×	×	×	–

APPENDIX B
PATCH BEFORE/AFTER SNIPPETS (P2 AND P4)

P2 (Travel expansion). Replace any non-adjacent hop with a sequence of legal edges.

```
// Before (illegal hop)
TRAVEL E3 Elmira->Bath

// After (per-edge sequence)
TRAVEL E3 Elmira->Corning
TRAVEL E3 Corning->Bath
```

P4 (CONVERT quantity). Ensure the quantity matches the number of loaded orange boxcars and attached empty tankers.

```
// Before (underspecified convert)
LOAD   E2 @Corning oranges cars=2
ATTACH E2 @Corning tanker qty=2
TRAVEL E2 Corning->Elmira
CONVERT E2 @Elmira

// After (explicit quantity)
LOAD    E2 @Corning oranges cars=2
ATTACH  E2 @Corning tanker qty=2
TRAVEL  E2 Corning->Elmira
CONVERT E2 @Elmira qty=2
```

APPENDIX C
UNSOLVED PROBLEMS: IMPOSSIBILITY NOTES

- **3-C (single-engine, five units).** With a three-loaded-car limit, a single engine must perform multiple trips. Deadlines and resource repositioning render the combined banana/juice requirement infeasible without lookahead and staged caching of rolling stock.
- **3-D (conversion conflict).** Simultaneous delivery of oranges *and* juice to the same deadline requires both preserving some oranges and converting others with limited tankers; local repair cannot satisfy both constraints under the given timing.
- **3-F (mixed sequencing).** Single-engine mixed cargo to a shared destination requires interleaving loads with conversions and detachments. Capacity and travel-time bounds preclude a feasible sequence without global search/backtracking.

APPENDIX D
TOKEN AND STEP SUMMARY (FROM METRICS.CSV)

TABLE V
TOKEN AND STEP SUMMARY BY CONFIGURATION (MEANS ACROSS 18 TASKS). WALL-CLOCK OMITTED DUE TO DETERMINISTIC CACHING.

Configuration	Tokens In	Tokens Out	Total Tokens	Steps
Baseline	717	280	997	9.3
Strategic	1017	350	1367	11.7
Patches	717	380	1097	12.7
Full	1017	393	1410	13.1

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.

Geng, Saibo, Cooper, Hudson, Moskal, Michał, Jenkins, Samuel, Berman, Julian, Ranchin, Nathan, West, Robert, Horvitz, Eric, and Nori, Harsha (2025). "Generating Structured Outputs from Language Models: Benchmark and Studies". In: *arXiv:2501.10868*. URL: https://arxiv.org/abs/2501.10868.

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

Koo, Terry, Liu, Frederick, and He, Luheng (2024). "Automata-based Constraints for Language Model Decoding". In: *arXiv:2407.08103*. URL: https://arxiv.org/abs/2407.08103.

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

Liu, C. L. and Layland, James W. (1973). "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". In: *Journal of the ACM* 20.1, pp. 46–61. URL: https://dl.acm.org/doi/10.1145/321738.321743.

Liu, Xiao et al. (2023). "AgentBench: Evaluating LLMs as Agents". In: *arXiv:2308.03688*. URL: https://arxiv.org/abs/2308.03688.

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*.

Zhou, Shuyan et al. (2023). "WebArena: A Realistic Web Environment for Building Autonomous Agents". In: *arXiv:2307.13854*. URL: https://arxiv.org/abs/2307.13854.