SED PUZZLE REASONING BY LLMs

Programming Task Submission

Precog @ IIT Hyderabad– 2025

Submitted by

Mohammed Abdul Rafe Sajid

3rd year BE-IT student at

Chaitanya Bharathi Institute of Technology (CBIT)

Hyderabad

GitHub Repository:

RepoLink

(see README.md to see summarized version of this report)

This report documents the methodology, experiments, evaluation metrics, and qualitative analysis conducted as part of the Precog programming task.

Table of Contents

Section 1: What I Did and Did Not Do

This section explicitly outlines the components of the task that were completed, partially completed, or not attempted, along with the rationale behind the

What I Did

- Designed and curated a custom dataset of sed-style string rewriting puzzles, ensuring all puzzles were solvable by construction.
- Implemented both auto-generated puzzles and manually curated adversarial puzzles to study different reasoning behaviors.
- Evaluated multiple prompting strategies on the dataset:
    - Zero-shot prompting
    - Few-shot prompting
    - Chain-of-Thought (CoT) prompting
    - Creative prompting using Constraint-Aware State Verification (CASV)
- Conducted experiments using multiple LLMs accessed through web interfaces (Gemini and GPT).
- Carefully documented all prompts used, model outputs, and failure cases.
- Designed custom evaluation metrics (Progress Score, Valid Transition Ratio, and a combined metric) to quantify partial correctness and reasoning quality beyond binary accuracy.
- Performed qualitative analysis comparing human problem-solving behavior with LLM behavior (Man vs Machine).
- Recorded observations, failure modes, and insights for each prompting strategy.

What I Did Not Do

- I did not use API access or large-scale automated benchmarking, due to rate limits, cost considerations, and setup overhead.
- I did not attempt exhaustive evaluation over the entire dataset; instead, I selected representative subsets to allow deeper qualitative analysis.

Reason for These Choices

The primary objective of this task was to analyze reasoning behavior, not to maximize raw performance metrics. Given this goal, I prioritized:

- depth of analysis over scale,
- careful documentation over automation,
- and interpretability of results over quantity.

All decisions regarding dataset size, prompting methods, and evaluation strategy were made with these priorities in mind and within the practical constraints of time and compute.

Section 2: Introduction & Problem Statement

Large Language Models (LLMs) have shown impressive performance on a wide range of natural language tasks. However, understanding how these models reason, especially in structured, rule-based settings, remains an open research question. This project focuses on studying LLM reasoning behavior using a class of problems inspired by the Unix sed tool, commonly referred to as string rewriting puzzles.

In these puzzles, a model is given an initial string along with a fixed set of rewrite rules (transitions). Each transition replaces all occurrences of a specified source substring with a target substring. The objective is to apply a sequence of transitions that reduces the initial string to the empty string. Importantly, transitions may have non-monotonic effects: some reduce the string length, while others expand or restructure it. This makes naive greedy strategies unreliable and requires careful planning.

Such puzzles provide a controlled and interpretable environment to analyze reasoning capabilities. Unlike natural language tasks, correctness here depends on strict rule application, state tracking, and global planning. A single incorrect transition can invalidate all future steps, making these puzzles particularly well-suited for studying failure modes such as hallucination, loss of state awareness, and locally optimal but globally incorrect reasoning.

The primary goal of this project is not to achieve perfect accuracy, but to systematically evaluate how different prompting strategies influence LLM behavior on these puzzles. Specifically, the project aims to answer the following questions:

- How do different prompting techniques affect the ability of LLMs to maintain state and apply valid transitions?
- Where do LLMs fail when puzzles require long-horizon planning or delayed decisions?
- Can explicit constraints or structured prompts reduce common reasoning errors?
- How can partial progress be meaningfully evaluated when full solutions are not found?

To address these questions, multiple prompting strategies were evaluated, including zero-shot, few-shot, Chain-of-Thought prompting, and a creative constraint-aware prompting method. In addition, custom evaluation metrics were designed to quantify not only final success, but also intermediate reasoning quality and progress toward the solution.

Section 3: Dataset Construction

To study LLM reasoning behaviour in a controlled manner, a custom dataset of sed-style string rewriting puzzles was constructed. Rather than relying on existing benchmarks, the dataset was designed from scratch to allow fine-grained control over puzzle structure, difficulty, and failure modes.

3.1 Puzzle Generation Strategy

All auto-generated puzzles were constructed using a reverse-generation approach to guarantee solvability. Instead of randomly sampling transitions and checking whether a solution exists, a valid sequence of transitions was first applied starting from an empty string, and the resulting string was treated as the puzzle's initial string. The solution sequence was then hidden, and the transitions were shuffled.

This approach ensures that:

- every generated puzzle has at least one valid solution,
- models are not penalized for unsolvable or ill-formed data,
- failures can be attributed to reasoning errors rather than dataset noise.

The code is as follows:

src > generate_puzzles.py > generate_puzzle

```python
import random
import json
import string
def random_token(min_len=2, max_len=4):
    length = random.randint(min_len, max_len)
    return ''.join(random.choices(string.ascii_uppercase, k=length))
def generate_puzzle(problem_id, difficulty="easy"):
    transitions = []
    current_string = ""

    if difficulty == "easy":
        steps = random.randint(2, 3)
    elif difficulty == "medium":
        steps = random.randint(4, 6)
    else:  # hard
        steps = random.randint(6, 8)
    inserted_tokens = []
    for _ in range(steps):
        token = random_token()
        inserted_tokens.append(token)
        current_string += token
        transitions.append({
            "src": token,
            "tgt": ""
        })

    # ADD DISTRACTORS HERE
    transitions = add_distractors(transitions, inserted_tokens)

    # Shuffle transitions to make reasoning non-trivial
    random.shuffle(transitions)
```

```python
    # ADD DISTRACTORS HERE
    transitions = add_distractors(transitions, inserted_tokens)

    # Shuffle transitions to make reasoning non-trivial
    random.shuffle(transitions)

    return {
        "problem_id": f"{problem_id:03d}",
        "initial_string": current_string,
        "transitions": transitions,
        "difficulty": difficulty
    }

def add_distractors(transitions, tokens):
    distractors = []
    for t in tokens:
        if len(t) > 2:
            distractors.append({
                "src": t[:2],
                "tgt": t[1:]
            })
    return transitions + distractors

def generate_dataset(n=100):
    dataset = []
    difficulties = ["easy", "medium", "hard"]

```

```python
def generate_dataset(n=100):
    dataset = []
    difficulties = ["easy", "medium", "hard"]

    for i in range(n):
        difficulty = random.choice(difficulties)
        puzzle = generate_puzzle(i, difficulty)
        dataset.append(puzzle)

    return dataset

if __name__ == "__main__":
    random.seed(42)

    puzzles = generate_dataset(100)

    with open("data/puzzles.json", "w") as f:
        json.dump(puzzles, f, indent=2)

    print("Generated 100 puzzles")

```

3.2 Difficulty Levels

Puzzles were categorized into three difficulty levels based on the number of transitions and the complexity of dependencies between them:

- **Easy:**
  2–3 transitions with non-overlapping substrings and largely independent deletions.

- **Medium:**
  4–7 transitions, including moderate dependencies and ordering constraints.
- **Hard:**
  Puzzles requiring longer transition chains and careful ordering.

During experimentation, it was observed that auto-generated hard puzzles were often long but not genuinely difficult. To address this, a separate set of manually curated hard puzzles was created.

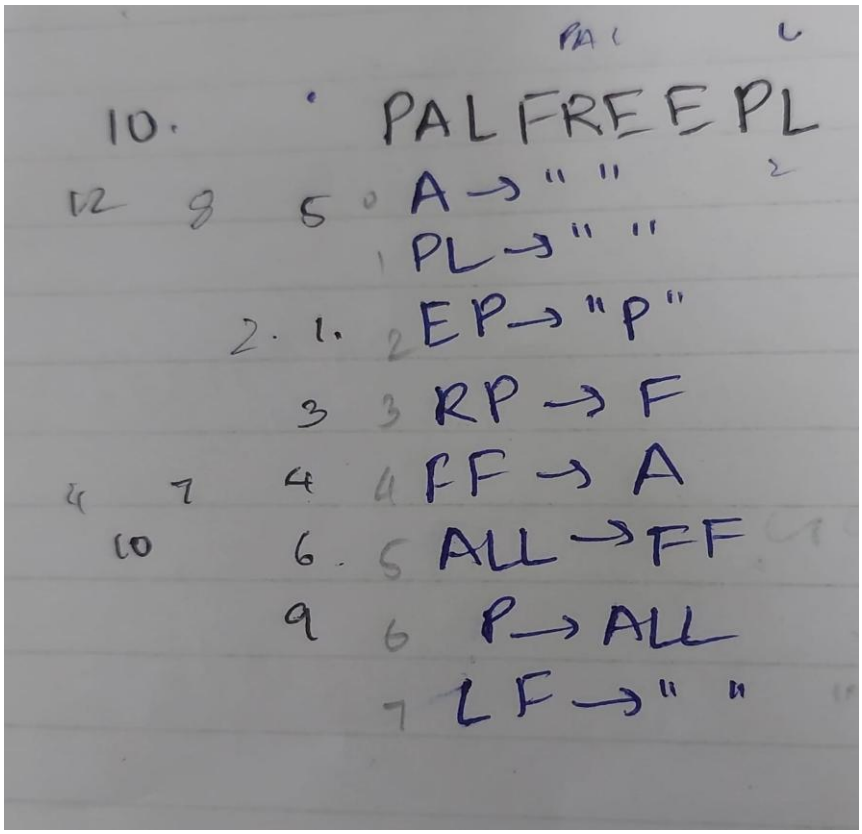
3.3 Manually Curated Adversarial Puzzles

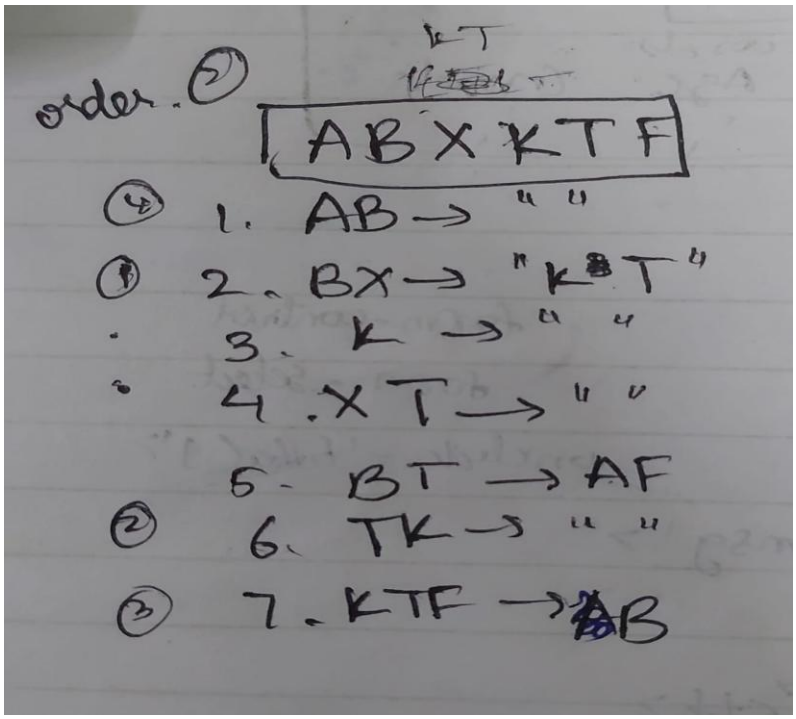
To better evaluate long-horizon reasoning and global planning, 10 hard puzzles were manually designed. These puzzles intentionally include:

- overlapping source substrings,
- transitions that expand the string before reducing it,
- traps where early deletions invalidate future steps.

These puzzles typically require 7–15 minutes for humans to solve and were found to be significantly more challenging for LLMs than auto-generated puzzles of similar length.

Some of them are:





3.4 Distractor Transitions

In addition to solution-preserving deletion rules, distractor transitions were introduced. These transitions are syntactically valid but:

- partially rewrite substrings instead of deleting them,
- introduce loops or dead ends,
- appear locally useful but are globally harmful.

The inclusion of distractors forces models to reason about transition ordering and long-term consequences, rather than greedily applying locally valid rules.

3.5 Dataset Size and Scope

The final dataset consists of 110 puzzles:

- 100 auto-generated puzzles
- 10 manually curated adversarial puzzles

The dataset size was intentionally kept small to prioritize depth of analysis and careful inspection of model behavior over large-scale benchmarking, which aligns with the exploratory nature of this task.

Example records in puzzles.json:

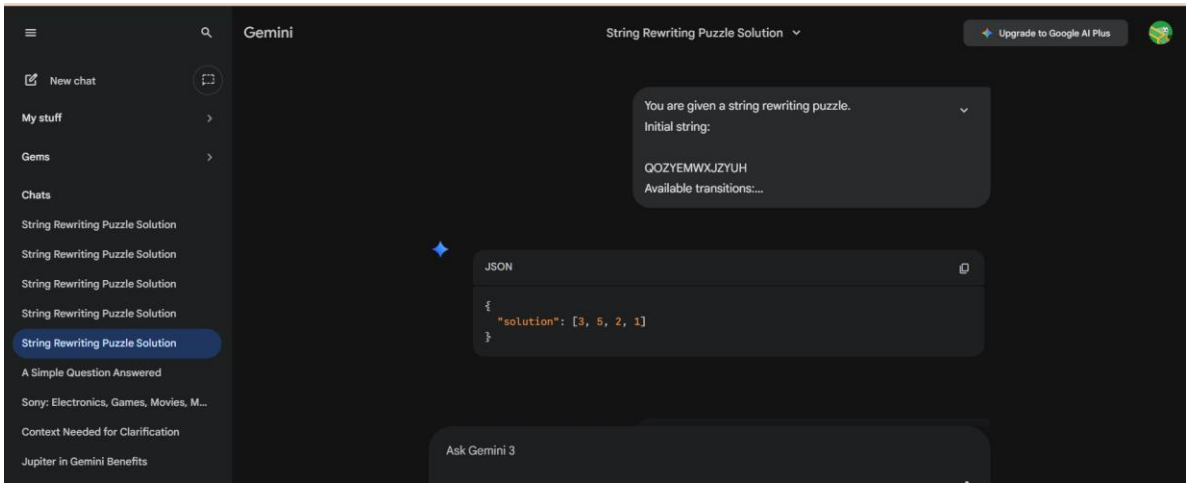
```
 {
   "problem_id": "005",
   "initial_string": "PFASFX",
   "transitions": [
     {
       "src": "FX",
       "tgt": ""
     },
     {
       "src": "PF",
       "tgt": ""
     },


{
       "src": "AS",
       "tgt": ""
     }
   }
```

Section 4: Prompting Strategies Evaluated

To understand how different prompting techniques influence LLM reasoning on string rewriting puzzles, multiple prompting strategies were evaluated. Each strategy was tested on a small but representative subset of puzzles, and all prompts, outputs, and failure cases were carefully documented.

The focus of this evaluation was not on maximizing success rate, but on analyzing reasoning behavior, failure modes, and state consistency under different prompting conditions.



4.1 Zero-shot Prompting

In zero-shot prompting, the model was provided only with the puzzle description and asked to find any valid sequence of transitions that reduces the initial string to the empty string. No examples or intermediate guidance were provided.

The prompt specified the initial string, a list of indexed transitions, and the required JSON output format. Models were instructed that transitions could be reused and that the goal was not to minimize the number of steps.

Experimental Setup
Zero-shot prompting was evaluated on:

- 3 easy puzzles (auto-generated),
- 4 medium puzzles (auto-generated),
- 3 hard puzzles (manually curated).

Each puzzle was attempted exactly once, with no retries or corrections.

The exact prompt given was:

You are given a string rewriting puzzle.

Initial string:

<I have kept the string here>

Available transitions:

Each transition replaces ALL occurrences of src with tgt.

Transitions (indexed):

0. "src1"->"tgt1"

1. "src2" -> "tgt2"

... <I have replaced the real transitions here>

Your task:

Return ANY sequence of transition indices that reduces the initial string to an empty string.

Rules:

Transitions can be reused.

The goal is not to minimize steps.

Return only a JSON object in the following format:

```
{
  "solution": [index1, index2, ...]
}
```

Quantitative Results

Difficulty	Solved	Total
Easy	3	3
Medium	3	4
Hard	0	3

Observations
Zero-shot prompting performed reliably on easy puzzles and most medium puzzles. Failures began to appear when transitions interacted non-trivially or required careful ordering. On manually curated hard puzzles, zero-shot prompting failed consistently.

Failure Modes
Common failure modes included:

- applying transitions whose source substrings were no longer present,
- planning sequences that appeared locally correct but were globally invalid,
- committing to early deletions that prevented future progress.

Key Insight
Zero-shot prompting is sufficient for surface-level string rewriting but struggles to maintain global state consistency in adversarial settings.

4.2 Few-shot Prompting

Few-shot prompting extended the zero-shot setup by providing a single solved example before the target puzzle. The example demonstrated a valid sequence of transitions without explicitly explaining the reasoning steps.

The goal was to provide an inductive bias toward correct transition ordering and state tracking.

Experimental Setup
Few-shot prompting was evaluated on:

- 2 medium puzzles (auto-generated),
- 3 hard puzzles (manually curated).

Prompts were tested on both Gemini and GPT models, and differences in model behavior were explicitly recorded.

The exact prompt given was:

You are given string rewriting puzzles.

Each puzzle has:

- an initial string

- a list of indexed transitions

Each transition replaces ALL occurrences of src with tgt.

--- Example ---

Initial string:

PFASFX <for examople>

Transitions (indexed):

0: FX -> ""

1: PF -> ""

2: AS -> ""

One valid solution is:

[2, 1, 0]

--- New Puzzle ---

Initial string:

<NEW_INITIAL_STRING>

Transitions (indexed):

0: ...

1: ...

2: ...

...

Your task:

Return ANY sequence of transition indices that reduces the initial string

to an empty string.

Rules:

- Transitions can be reused.

- The goal is not to minimize steps.

- Return only a JSON object in this format:

```
{
  "solution": [index1, index2, ...]
}
```

Quantitative Results

Difficulty	Solved	Total
Medium	2	2
Hard	1	6

- Hard ones's in detail

Puzzle	Model	Solved?	Failure type
H02	Gemini-3	No	Invalid transition
H02	GPT Go	Yes	solved after taking time
H04	Gemini-3	No	Invalid transition
H04	GPT Go	No	Invalid transition
H07	Gemini-3	No	Invalid transition
H07	GPT Go	No	Invalid transition

Observations

Few-shot prompting improved performance on medium puzzles compared to zero-shot prompting. In one case, a medium puzzle that failed under zero-shot was successfully solved

using few-shot prompting. However, performance on hard puzzles remained unreliable and model-dependent.

Failure Modes
Failures typically involved:

- hallucinated applicability of transitions,
- loss of state awareness after several correct steps,
- reaching non-empty terminal strings with no valid transitions remaining.

Key Insight
Few-shot prompting provides useful guidance for simpler instances but does not reliably address long-horizon planning or adversarial dependencies.

4.3 Chain-of-Thought (CoT) Prompting

In Chain-of-Thought prompting, models were explicitly instructed to reason step-by-step, track the current string after each transition, and avoid applying transitions whose preconditions were not satisfied.

The prompt required models to show intermediate reasoning but output only the final solution in JSON format.

Experimental Setup
CoT prompting was evaluated on:

- 1 medium puzzle (auto-generated),
- 2 hard puzzles (manually curated).

Each puzzle was attempted once per model.

The exact prompt given was:

You are given a string rewriting puzzle.

Initial string:

<INITIAL_STRING>

Transitions (indexed):

0: src -> tgt

1: src -> tgt

2: src -> tgt

...

Task:

Find a sequence of transitions that reduces the initial string to the empty string.

Instructions:

- Think step by step.

- After each transition, explicitly write the current string.

- Do not apply a transition unless its src is present in the current string.

- If no valid transition applies, state that clearly.

- At the end, output only the final solution in this JSON format:

```
{
  "solution": [index1, index2, ...]
}
```

Quantitative Results

Difficulty	Solved	Total
Medium	1	1
Hard	0	2

Observations
CoT prompting successfully solved the medium puzzle and significantly reduced random or hallucinated steps. Models demonstrated improved discipline in checking applicability and were more likely to stop and admit uncertainty.

However, CoT prompting did not succeed on hard puzzles requiring global planning or backtracking.

Failure Modes
Observed failures included:

- incorrect application of transitions despite explicit reasoning,
- premature termination with non-empty strings,
- incorrectly concluding that no solution exists for solvable puzzles.

Key Insight
Chain-of-Thought improves reasoning transparency and local consistency but is insufficient for solving adversarial puzzles that require long-term planning and recovery.

4.4 Creative Prompting: Constraint-Aware State Verification (CASV)

For the creative component of this task, a **Constraint-Aware State Verification (CASV)** prompting strategy was designed. This approach directly targets the dominant failure mode observed in earlier experiments: applying transitions when their source substrings were not present.

In CASV prompting, the model was instructed to:

- explicitly list all applicable transitions at each step,
- select a transition only from this list,
- update the string after each step,
- stop immediately if no valid transition exists.

Experimental Setup
CASV prompting was evaluated on:

- 1 medium puzzle (auto-generated),
- 4 hard puzzles (manually curated).

Each puzzle was tested on both Gemini and GPT models under identical conditions.

The exact prompt given was:

You are given a string rewriting puzzle.

Initial string:

<INITIAL_STRING>

Transitions (indexed):

0: src -> tgt

1: src -> tgt

2: src -> tgt

...

Task:

Find a sequence of transitions that reduces the initial string to the empty string.

Special Instructions:

- At each step, list all transitions whose src is currently present in the string.

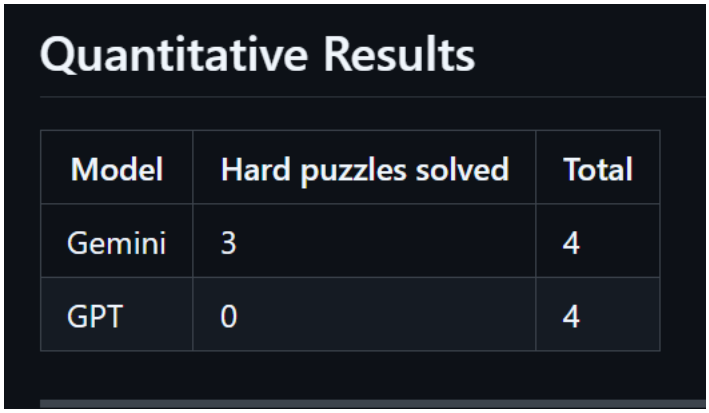
- Choose one valid transition from that list.

- Apply it and update the string.

- If no transition is applicable at any step, explicitly state that and stop.

At the end, output only the final solution in this JSON format:

```
{
  "solution": [index1, index2, ...]
}
```



Observations
CASV prompting led to a clear improvement in performance on hard puzzles for Gemini, which successfully solved multiple adversarial instances that previous prompting strategies could not. The prompt significantly reduced hallucinated steps and invalid transition applications.

GPT models, however, struggled to benefit from the same constraints and frequently failed due to subtle substring confusions or premature termination.

Failure Modes
Even under CASV prompting, some failures persisted:

- late-stage invalid transition application after several correct steps,
- confusion between similar substrings,

- stopping with a non-empty string despite valid transitions remaining.

Key Insight
Explicitly enforcing transition applicability can substantially improve reasoning on adversarial puzzles, but the effectiveness of this strategy is strongly model-dependent.

Storing all solution in .json

Example:


```
{
  "problem_id": "099",
  "difficulty": "medium",
  "predicted_solution":[8, 4, 1, 2, 7, 6],
  "solved": true,
  "final_string": ""
},
{
  "problem_id": "H02",
  "difficulty": "hard",
  "model":"Gemini-3",
  "predicted_solution":[6, 1, 5, 0],
  "solved": false,
  "final_string": "AKTB",
"failure_reason": "transition 5 (TK→'') not applicable"
```

Section 5: Evaluation Metrics

Evaluating model performance on string rewriting puzzles using only binary success or failure is insufficient. Many model outputs fail to fully solve a puzzle but still demonstrate meaningful partial reasoning, such as applying several correct transitions before making a mistake. To capture this behaviour, custom evaluation metrics were designed to measure both progress toward the solution and reasoning consistency.

5.1 Progress Score (P)

The first metric measures how close a model gets to solving a puzzle, based purely on string length reduction.

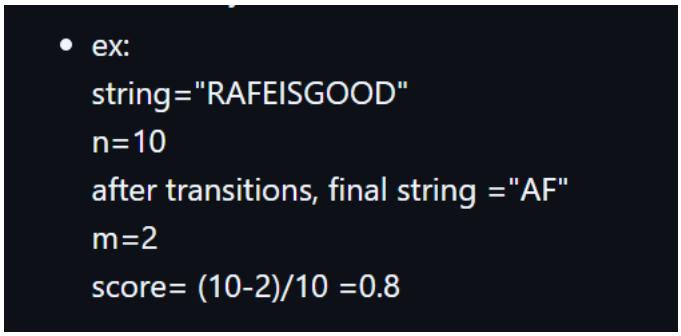
Let:

- n be the length of the initial string,
- m be the length of the final string after applying the model's proposed transitions.

The **Progress Score (P)** is defined as:

$$P = \frac{n - m}{n}$$

This metric takes values in the range $(-\infty, 1]$, where values closer to 1 indicate solutions closer to completion.



- A solved puzzle has P = 1.
- If the final string is longer than the initial string, P becomes negative, indicating harmful reasoning.

This metric captures **how much progress** was made but does not account for whether transitions were applied validly.

5.2 Limitations of Progress Score

While Progress Score is intuitive, it has several limitations:

- It does not account for the structure or length of transition source substrings.
- A puzzle may be one transition away from completion but still receive a moderate score if that transition has a long source.
- Incorrect transitions may reduce string length accidentally, inflating the score despite incorrect reasoning.

These limitations motivate the need for a complementary metric.

5.3 Valid Transition Ratio (VTR)

To measure reasoning discipline, the **Valid Transition Ratio (VTR)** was introduced.

VTR is defined as:

$$VTR = \frac{\text{Number of valid transitions applied}}{\text{Number of proposed transitions}}$$

Ex:
String ="ABCFK"
Transitions: 0. "AB"->""

1. "FK" -""
2. "CA" -> ""
3. "C" -> ""
4. "DK"->"C"

Suppose Solution is [1,0,2,4,3]
Proposed transitions: 5

1 is applicable, now str= "ABC"
0 is applicable, now str="C"
2 is not applicable, here we will STOP

Applicable Transitions: 2

therefore, VTR = 2/5 = 0.4

Execution is simulated step by step and **stops at the first invalid transition**. All transitions after the first invalid step are ignored.

- If the proposed solution is empty ([]), VTR is defined as 0, indicating no demonstrated reasoning.
- If all proposed transitions are valid, VTR equals 1.

This metric captures **how long a model maintains logical consistency** during execution.

5.4 Limitations of VTR

VTR also has limitations:

- It does not measure how much progress is made toward the final solution.
- Longer solution attempts are penalized more heavily than short conservative ones.
- It does not account for recovery after an invalid transition, as execution halts immediately.

5.5 Combined Metric

To balance the strengths and weaknesses of both metrics, a combined score is proposed:

$$m = P \times VTR$$

This combined metric rewards solutions that both:

- make substantial progress toward the solution, and
- maintain valid reasoning for a longer prefix of the proposed sequence.

The maximum possible value of m is 1.
A negative value of m indicates that the model expanded the string overall despite partially valid transitions.

5.6 Metric Validation Through Examples

The proposed metric was evaluated on representative examples from zero-shot, few-shot, and Chain-of-Thought prompting. These examples demonstrate that the combined metric assigns higher scores to solutions that are both closer to completion and more logically consistent, while penalizing early failures and harmful reasoning.

Initial string: `LKFTNTBOORK`
Predicted solution: `[1, 0, 2, 14, 9]`

All proposed transitions are valid, but the final string is non-empty.
Final string obtained: `K`

Metric computation

- Initial length, `n = 11`
- Final length, `m = 1`

Progress Score (P):
P = (n - m) / n = (11 - 1) / 11 ≈ 0.91

Valid Transition Ratio (VTR):
Valid transitions = 5
Proposed transitions = 5

VTR = 5 / 5 = 1

Final Metric:
m = P x VTR ≈ 0.91 x 1 ≈ 0.91

This represents a near-complete solution with correct reasoning but incomplete termination.

Section 6: Experimental Results & Analysis

This section consolidates results across all prompting strategies and models, focusing on **patterns of success and failure** rather than isolated outcomes. The analysis emphasizes *why* certain approaches succeed or fail, in line with the evaluation criteria outlined in the task description.

6.1 Performance Across Prompting Strategies

A clear progression in reasoning quality is observed as prompting becomes more structured.

- **Zero-shot prompting** performs well on easy and many medium puzzles, where solution paths are largely monotonic and transitions do not interact adversarially. However, it fails consistently on manually curated hard puzzles.
- **Few-shot prompting** improves performance on medium puzzles by providing a weak inductive bias toward correct transition ordering. This improvement does not reliably extend to hard puzzles.

- **Chain-of-Thought prompting** reduces hallucination and random guessing by encouraging explicit state tracking. While this leads to cleaner reasoning traces and closer partial solutions, it does not resolve global planning failures.
- **Constraint-Aware State Verification (CASV)** prompting produces the strongest results on adversarial puzzles, particularly for Gemini. By forcing explicit applicability checks, it reduces invalid transitions and enables longer correct reasoning prefixes.

6.2 Cross-Model Comparison

Model-dependent behavior was observed across all prompting strategies.

Gemini models benefited significantly from increased structure in prompts. Under CASV prompting, Gemini was able to solve multiple manually curated hard puzzles that were unsolved by other prompting methods.

GPT models, on the other hand, often failed to leverage the same constraints effectively. Common issues included confusion between similar substrings, premature termination, and incorrect rejection of valid transitions. This suggests differences in how models internalize and follow explicit procedural instructions.

6.3 Analysis of Failure Modes

Across all strategies, several recurring failure patterns emerged:

- **Loss of state awareness**: models apply transitions whose source substrings are no longer present.
- **Local over global reasoning**: early deletions that appear beneficial locally invalidate future necessary steps.
- **Late-stage collapse**: after several correct steps, a single invalid transition leads to failure.
- **Premature termination**: models stop with a non-empty string despite valid transitions remaining.

These failures highlight the difficulty LLMs face in maintaining long-horizon consistency under strict symbolic constraints.

6.4 Interpreting Metric Behavior

The proposed evaluation metric ($m = P \times VTR$) aligns well with qualitative observations:

- Solutions that fail early receive low scores due to both low progress and low VTR.

- Near-complete solutions with valid reasoning but incomplete termination receive high scores.
- Harmful reasoning that expands the string results in negative or near-zero scores.

This demonstrates that the metric captures meaningful differences between partial solutions that binary accuracy alone would miss.

6.5 Hypotheses on Success and Failure

Based on observed behavior, the following hypotheses are proposed:

- LLMs struggle with **global planning** and invariant preservation in adversarial symbolic tasks.
- Explicit constraints improve performance only when the model can reliably internalize and follow them.
- Failures are not due to lack of pattern recognition, but due to limited ability to reason about long-term consequences of actions.

These hypotheses are consistent with both the experimental results and the Man vs Machine analysis presented in the next section.

Section 7: Man vs Machine

This section presents a qualitative comparison between human problem-solving behavior and LLM behavior on string rewriting puzzles. The goal is not to measure speed or accuracy alone, but to understand **how** solutions are approached and **where reasoning diverges**.

7.1 Human Solving Behaviour

To better understand human reasoning, a subset of puzzles was shared with friends, family members, also through social platforms. They were asked to solve the puzzles, report the time taken, and describe where they felt difficulty.

Manually curated hard puzzles were generally solvable by humans within **5–10 minutes**, while auto-generated easy and medium puzzles were typically solved within **1–2 minutes**.

Human solvers exhibited several consistent behaviours:

- They tended to reason **globally**, thinking ahead about how current deletions might affect future possibilities.
- Many participants mentally tracked which substrings needed to be preserved until specific transitions became applicable.

- When forward reasoning failed, some solvers naturally switched to **backward reasoning**, working from the empty string toward the initial string.

These behaviours allowed humans to avoid early mistakes that would later block valid solutions.

7.2 LLM Solving Behavior

LLMs demonstrated a bit contrasting reasoning pattern.

On auto-generated easy and medium puzzles, LLMs solved problems quickly, often within seconds, due to strong **local pattern-matching abilities** and consistent rule application. However, on manually curated hard puzzles, LLMs frequently failed despite producing outputs rapidly.

Common LLM behaviors included:

- Committing to locally attractive deletions without considering long-term consequences.
- Losing track of global state after several correct steps.
- Failing to recover after an early mistake, even when explicitly instructed to verify applicability (as in CASV prompting).

While constraint-aware prompting improved performance for some models, these improvements were not consistent across all LLMs.

7.3 Key Insight

Although humans and LLMs differ in speed and reliability across puzzle types, no definitive conclusion can be drawn that extremely hard puzzles are solvable only by humans and not by machines.

Instead, a distinction in strengths emerges:

- Humans excel at **long-horizon planning**, global state tracking, and preserving necessary invariants.
- LLMs perform well on tasks dominated by **local and linear pattern matching**.

Adversarial puzzle design exposes this gap clearly, highlighting fundamental differences between human reasoning and current LLM behavior.

Section 8: Limitations & Future Work

While this project provides meaningful insights into LLM reasoning behavior on string rewriting tasks, several limitations remain.

8.1 Limitations

One key limitation of this work is the **scale of experimentation**. Due to practical constraints such as rate limits, cost considerations, and the use of web-based interfaces, large-scale automated benchmarking was not performed. Instead, a carefully selected subset of puzzles was evaluated to allow deeper qualitative analysis.

Additionally, it was observed that the evaluation metric assumes no recovery or backtracking after an invalid transition. Once an incorrect transition is applied, execution halts. While this matches the semantics of the puzzle setup, it restricts the evaluation of recovery-oriented reasoning strategies.

Finally, the evaluation focused on a limited number of models. Although clear model-dependent trends were observed, broader conclusions would require testing across a wider range of architectures and versions.

8.2 Future Work

With additional time and resources, several extensions could be explored.

Future work could involve:

- Implementing classical symbolic or search-based solvers as baselines to compare against LLM behavior.
- Exploring hybrid approaches, where LLMs generate candidate plans that are verified or corrected by symbolic execution.
- Investigating backtracking or planning-aware prompting strategies that allow models to revise earlier decisions.
- Scaling experiments using API-based access to evaluate performance across a larger portion of the dataset.
- Studying whether fine-tuning or reinforcement learning could help models internalize global planning constraints.

These directions could further clarify the gap between human and machine reasoning in structured symbolic tasks.

Section 9: External Resources & Attribution

This project was completed independently, with all Experimentations, Evaluation metrics formula, analysis, and conclusions performed by the myself. AI tools such as ChatGPT was used for discussion, writing this report (after providing all self-written READMEs and other required data), clarifying doubts, refining code and creating Prompts for specific problems using a template, which made my work more efficient.

General literature, blogs, and discussions on Large Language Models, prompting strategies, and reasoning behavior were referenced to understand better. These resources were used for conceptual understanding rather than direct implementation.

This work complies with the task requirement that the submission be the result of independent effort, with appropriate acknowledgment of external assistance.

10. Conclusion

This project explored how Large Language Models reason over sed-style string rewriting puzzles, a class of structured symbolic tasks that require strict rule application, state tracking, and long-horizon planning. By designing a custom dataset and systematically evaluating multiple prompting strategies, the work aimed to move beyond binary success metrics and gain deeper insight into model behavior.

The experiments demonstrate that while LLMs perform strongly on puzzles dominated by local and linear pattern matching, they struggle with adversarial instances that require global planning, delayed decisions, and invariant preservation. Increasing prompt structure—from zero-shot to few-shot and Chain-of-Thought—improves reasoning discipline but does not fully resolve these challenges. A creative constraint-aware prompting strategy (CASV) was found to significantly improve performance for certain models, highlighting the importance of explicit state verification, though its effectiveness remains model-dependent.

To better capture partial reasoning quality, custom evaluation metrics were proposed that measure both progress toward the solution and logical consistency. These metrics provide a more nuanced view of model performance than binary accuracy alone and align well with qualitative observations from the experiments.

Finally, a Man vs Machine analysis revealed complementary strengths between humans and LLMs. Humans excel at global planning and adaptive reasoning, while LLMs are efficient at local pattern matching and rule application. Adversarial puzzle design exposes this gap clearly, offering a useful lens for studying the limitations of current LLM reasoning.

Overall, this work emphasizes that understanding *how* models reason—and why they fail—is as important as whether they succeed. The findings suggest promising directions for future research combining structured prompting, symbolic verification, and hybrid human–machine reasoning approaches.