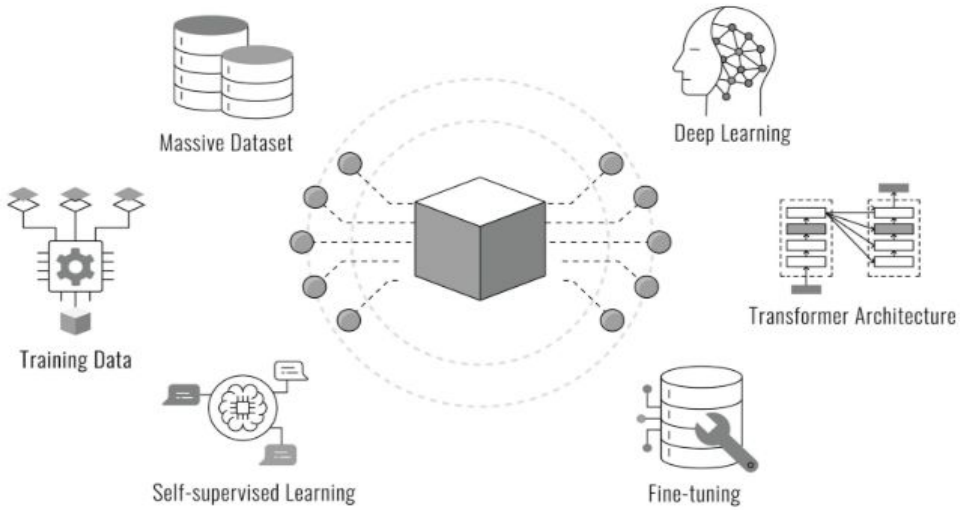
Prompt Engineering for Algorithmic Code Generation

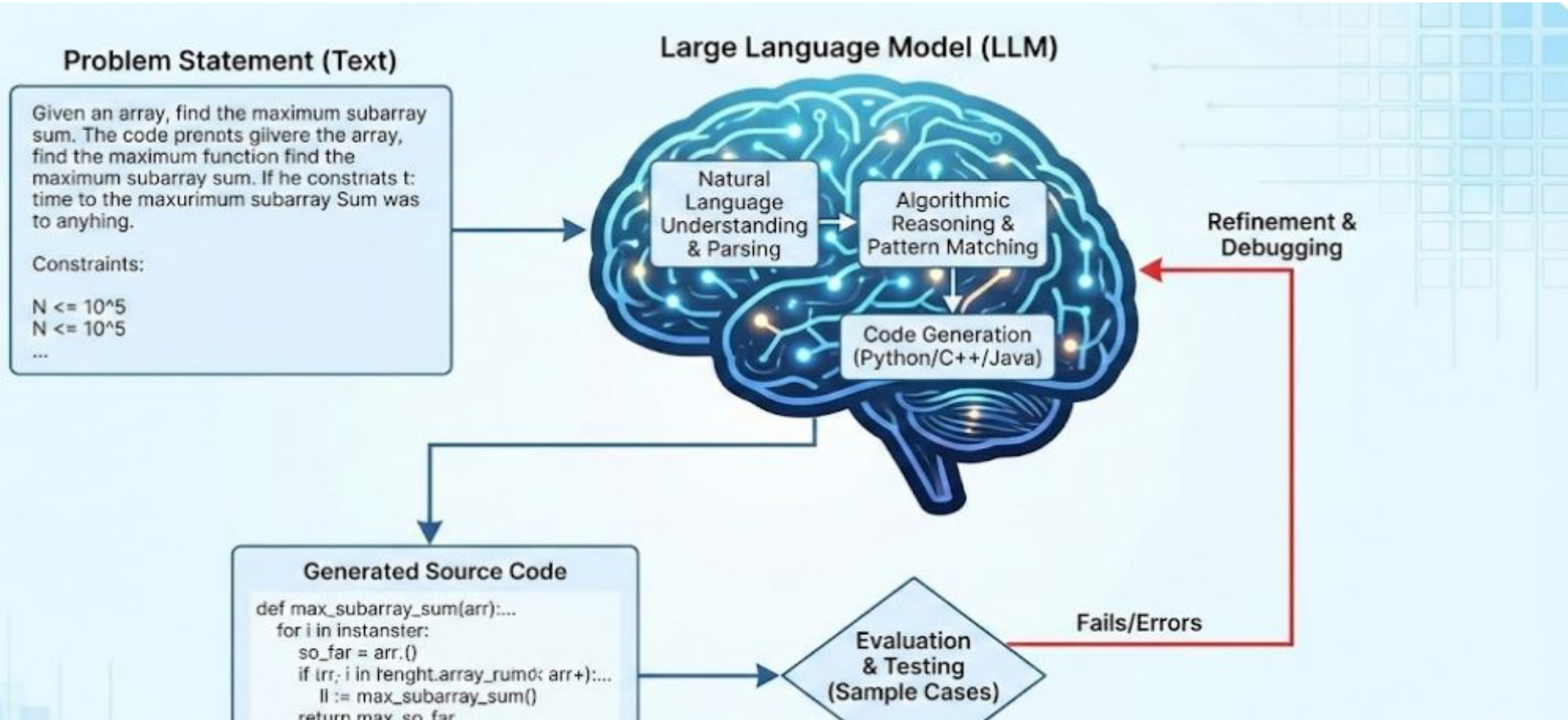
Comparative Analysis of Reasoning Strategies on LLM Performance

Team Members:

- Prakriti Adhikari
- Fiza Ashraf
- Aymen Noor



Context & Problem Definition



Context & Problem Definition

The Challenge:

- Large Language Models (LLMs) often **fail in highly reasoning-intensive domains** like competitive programming.
- These problems require **complex, multi-step logical deduction** and robust algorithm selection.

Project Task:

- Investigate and systematically compare the effect of **various prompt engineering strategies** on LLM performance when solving complex algorithmic problems.

Real-World Relevance:

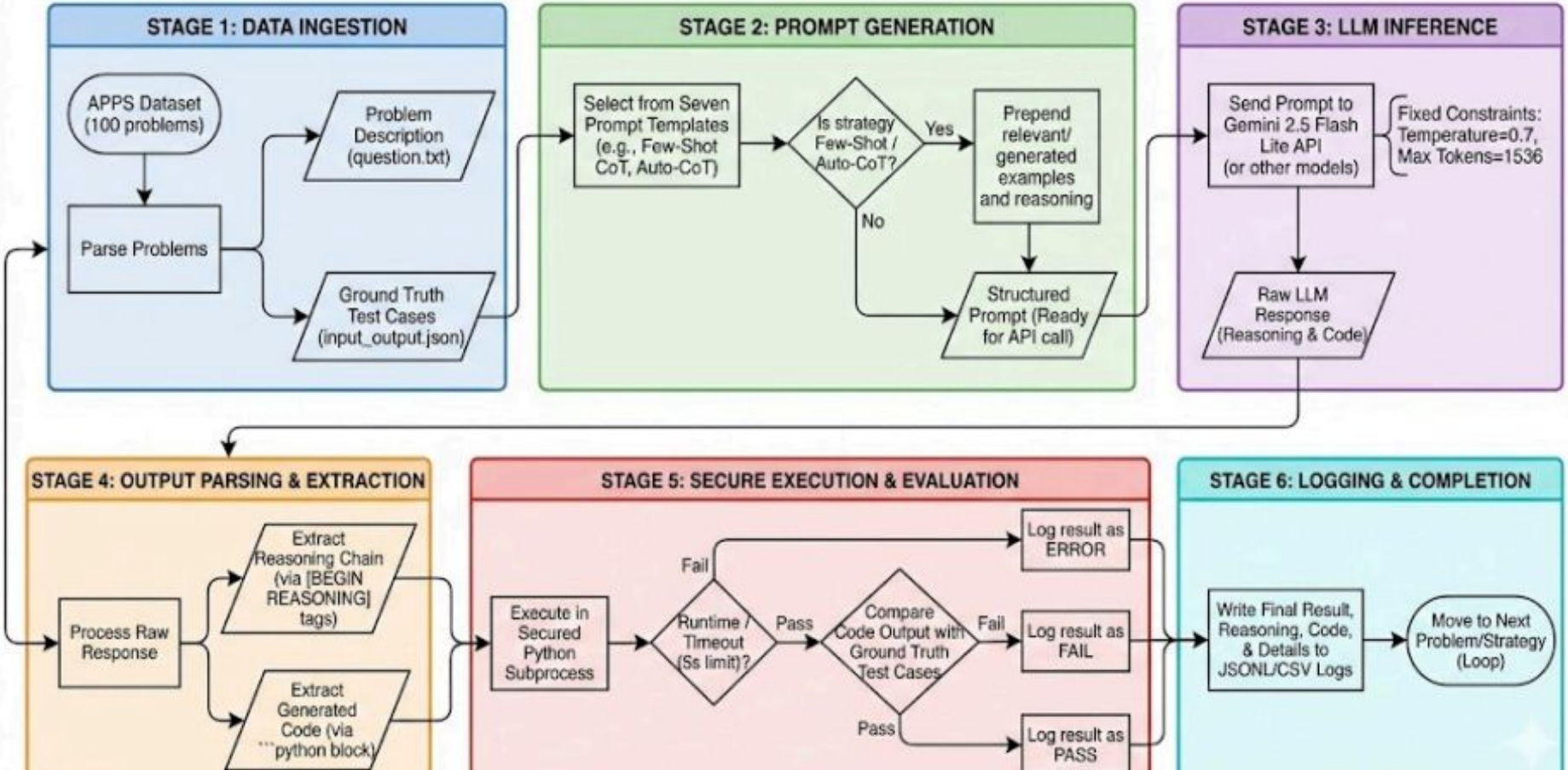
- Improving LLM capability for **advanced developer tooling**, automated code review, and high-stakes technical domains

Methodology - Pipeline & Dataset

Dataset:

- **APPS (Automated Programming Problem Set)** dataset.
- Subset: 100 problems, all tagged with **"Interview" difficulty**.
- Evaluation is **objective**, based on Ground Truth Test Cases.

System Design Flowchart:
EXPERIMENTAL PIPELINE: HIGH-THROUGHPUT LLM EVALUATION ON APPS DATASET



Experimental Pipeline (6 Stages):

1. **Data Ingestion:** Load problem descriptions and ground truth test cases.
2. **Prompt Generation:** Insert problem into one of **seven prompt templates**.
3. **LLM Inference:** Send prompt to primary model (**Gemini 2.5 Flash Lite API**).
4. **Output Parsing:** Extract reasoning ([BEGIN REASONING] tags) and code (python block).
5. **Secure Execution & Evaluation:** Execute code in a subprocess against test cases with a **strict 5s timeout**.
6. **Logging & Completion:** Log result as **PASS, FAIL, or ERROR**.

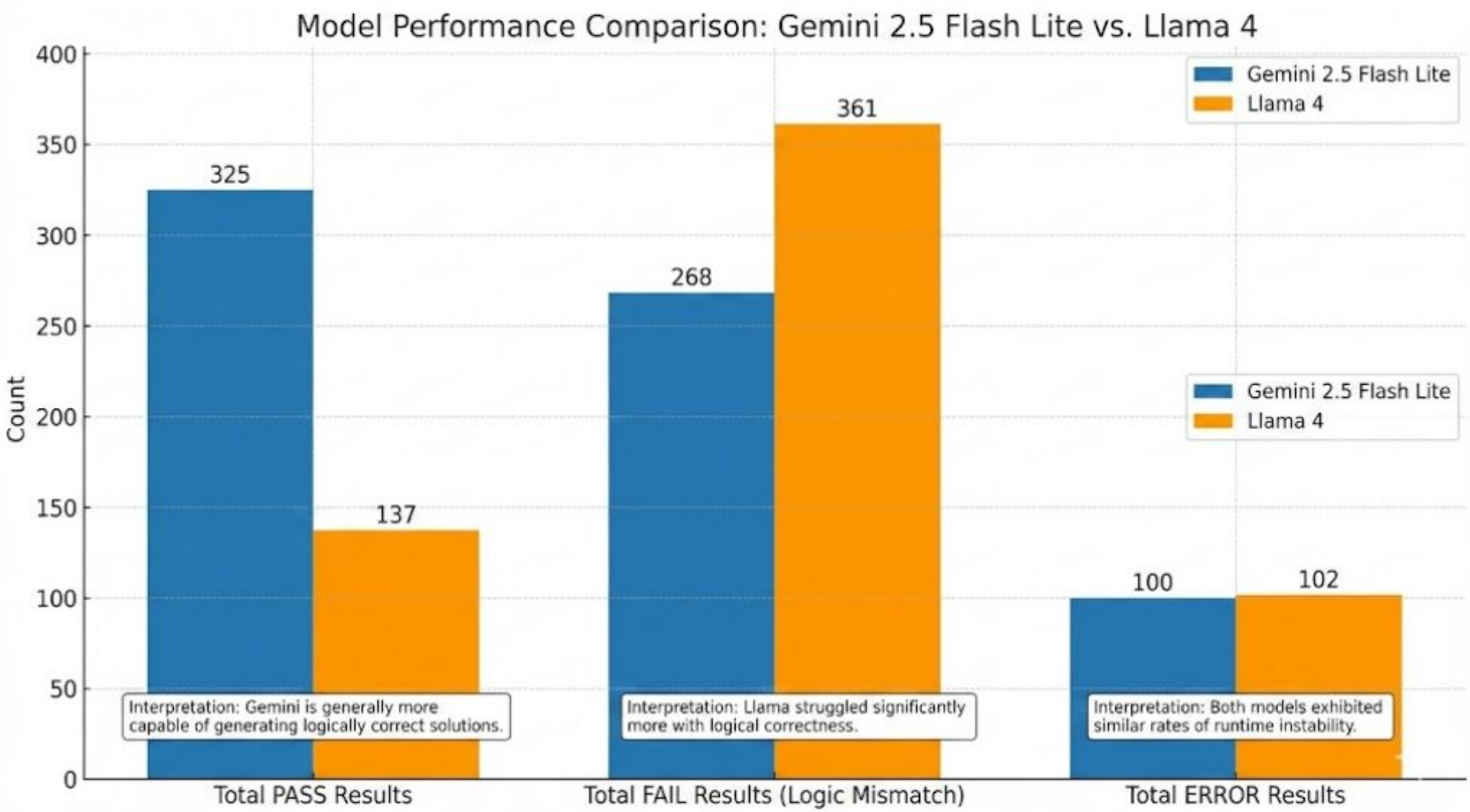
Prompting Strategies Compared

Strategy	Primary Goal	Key Feature
Zero-Shot	Assess inherent ability with mandated reasoning structure.	**No examples**. Mandates detailed reasoning steps.
Zero-Shot CoT	Zero-Shot variant with slightly different mandated steps.	Mandates steps: Restate problem, Identify constraints, Derive optimal solution.
Few-Shot	Evaluate benefit of in-context learning, isolating from reasoning.	Prepends **3 examples** (problem/solution), **no reasoning chains**.
Few-Shot CoT	Assess maximum benefit of **Gold Standard CoT**.	Prepends **3 examples** with full, high-quality **manual reasoning** and solution.
Auto-Few-Shot CoT	Evaluate scalability of **self-generated reasoning**.	LLM *first* generates reasoning for the 3 examples, which are then used.

Key Finding 1: Model Head-to-Head

Gemini 2.5 Flash Lite Consistently Outperformed Llama 4

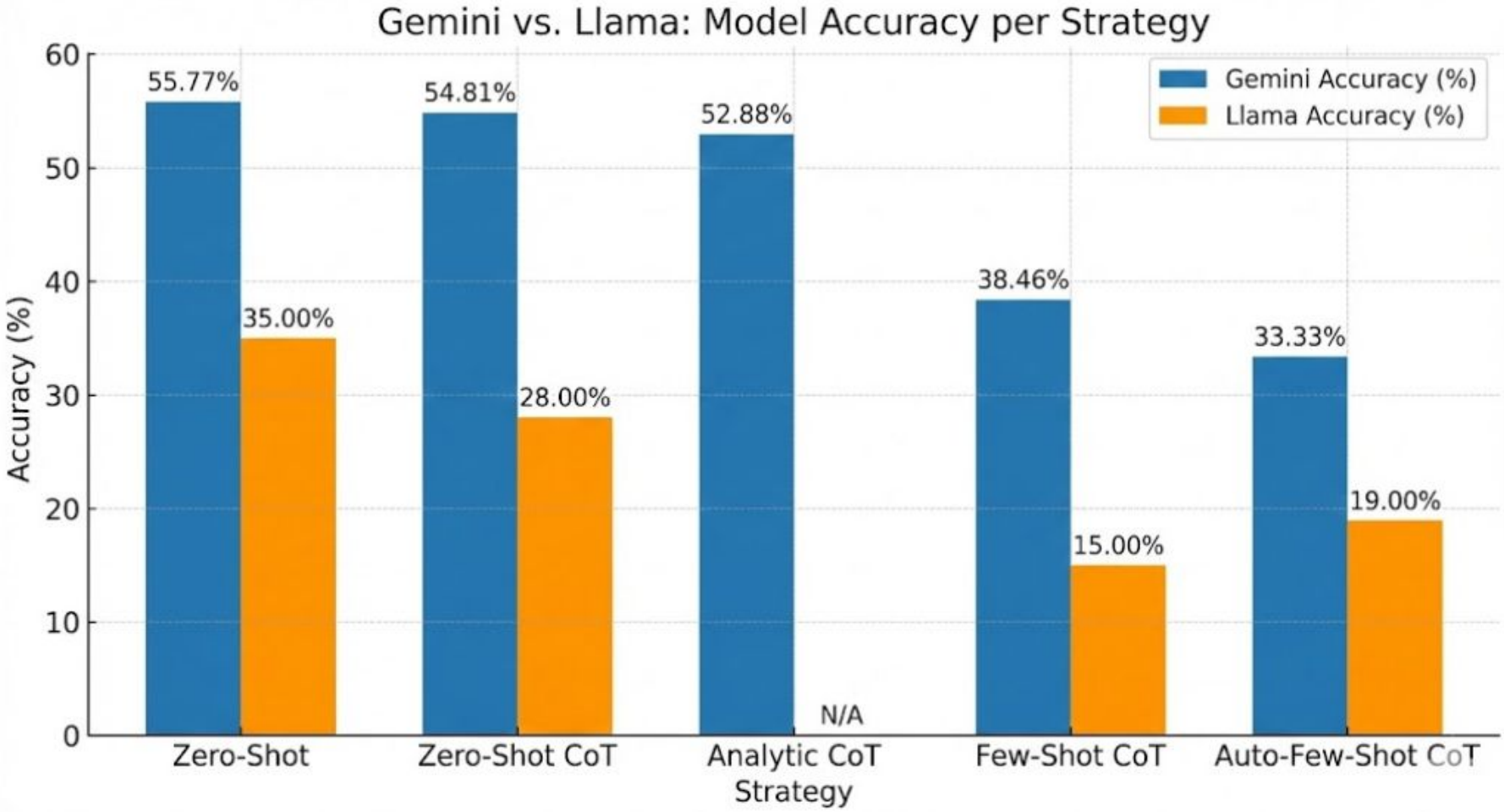
Metric	Gemini 2.5 Flash Lite (Total)	Llama 4 (Total)	Interpretation
Total PASS Results	**325**	137	**Gemini is generally more capable** of generating logically correct solutions.
Total FAIL Results (Logic Mismatch)	268	**361**	Llama struggled significantly more with **logical correctness**.
Total ERROR Results	100	102	Both models exhibited similar rates of runtime instability.


Model Performance Comparison: Gemini 2.5 Flash Lite vs. Llama 4
Count
400
350
300
250
200
150
100
50
0
Gemini 2.5 Flash Lite
Llama 4
325
137
268
361
100
102
Interpretation: Gemini is generally more capable of generating logically correct solutions.
Interpretation: Llama struggled significantly more with logical correctness.
Interpretation: Both models exhibited similar rates of runtime instability.
Total PASS Results
Total FAIL Results (Logic Mismatch)
Total ERROR Results

Key Finding 2: Strategy Effectiveness

Gemini: Simplicity Outperforms Complexity

Strategy	Gemini Accuracy (%)	Llama Accuracy (%)
Zero-Shot	**55.77**	35.00
Zero-Shot CoT	54.81	28.00
Analytic CoT	52.88	N/A
Few-Shot CoT	38.46	15.00
Auto-Few-Shot CoT	**33.33**	19.00


Gemini vs. Llama: Model Accuracy per Strategy
Gemini Accuracy (%)
Llama Accuracy (%)
Accuracy (%)
0
10
20
30
40
50
60
55.77%
35.00%
54.81%
28.00%
52.88%
N/A
38.46%
15.00%
33.33%
19.00%
Zero-Shot
Zero-Shot CoT
Analytic CoT
Few-Shot CoT
Auto-Few-Shot CoT
Strategy

Slide 6: Key Finding 2: Strategy Effectiveness

Interpretation for Gemini: Providing direct instructions and a structured analytical path (Zero-Shot CoT) was most effective.

Complex Few-Shot strategies (especially Auto-CoT) actively degraded performance due to context size issues and parsing variability.

Llama 4: CoT variants did not provide a clear, consistent accuracy advantage, and the model struggled severely with Few-Shot variants

Failure Analysis

Logic Mismatch (FAIL):

- **Overwhelming primary cause** of non-PASS outcomes for both models (Llama: 361; Gemini: 268).
- Confirms the challenge is in **algorithmic derivation and implementation accuracy**, not syntax.

Instability and Efficiency:

Timeout Errors: Suggests efficiency issues—code is correct but **too slow** for the 5-second limit (Llama: 36; Gemini: 21).

Auto-Few-Shot CoT was particularly unstable for Gemini, showing a **39% ERROR rate**, highlighting instability from model-generated training context.

- **Qualitative Insight:** Successful Zero-Shot CoT runs on Gemini often produced **concise, correct algorithmic reasoning** (e.g., using dynamic programming/prefix array), indicating the prompt successfully compels the articulation of the optimal approach.

Conclusion

Simpler prompting strategies consistently outperformed complex Few-Shot CoT variants for the high-performing model (Gemini).

Relying on the model's inherent, structured reasoning ability is more robust than introducing excessive, potentially error-prone context.

Limitations:

- Persistent dominance of **Mismatch (logic errors)**.
- **Strict 5-second timeout** led to performance issues from unoptimized solutions.
- Limited to 100 "Interview" difficulty problems.

Future Work:

- **Enhancing Logical Correctness:** Focus on **Self-Correction (Self-Refine)** mechanisms, where the model iteratively debugs its code using unit test feedback.
- **Model Fine-Tuning:** Fine-tuning a base model (e.g., Llama 4) on the APPS dataset to drastically improve logical correctness and output reliability.