



Generating Streamlining Constraints with Large Language Models

Florentina Voboril · Vaidyanathan P. R. · Stefan Szeider

Algorithms and Complexity Group
TU Wien, Vienna, Austria



Constraint Programming Recap

Key Concepts:

- Declarative constraints to model decision and optimization problems
- CP applications: scheduling, routing, and planning

```
int: n = 5;  
array[1..n] of var 1..n: queens;  
constraint alldifferent(queens);  
constraint forall(i in 1..n, j in i+1..n)  
    abs(queens[i] - queens[j]) != abs(i - j)
```

Importance:

- Quality of CP models directly impacts solving performance

The strength of Constraint Programming lies in its declarative nature, which allows for concise problem definitions that directly influence the efficiency and effectiveness of the solving process.

Introduction to Streamliners

Definition:

- Streamliners are additional constraints used to narrow the search space (Gomes and Sellmann, 2004)

Mechanism:

- Incorporate domain-specific insights to prune unpromising search areas
- Not necessarily sound; unlike symmetry-breaking constraints (Flener et al., 2001)

Context:

- Traditional streamliner generation versus LLM-based methods

Streamliners are defined by their ability to reduce the solution space through informed constraints, distinguishing them from standard symmetry-breaking techniques.

```
int: n = 5;  
array[1..n] of var 1..n: queens;  
constraint alldifferent(queens);  
constraint forall(i in 1..n, j in i+1..n)  
    abs(queens[i] - queens[j]) != abs(i - j)
```

```
constraint queens[1] <= ceil(n/2);
```

Traditional Streamliner Generation

Manual & Systematic Methods:

- Crafted by experts or generated through exhaustive offline testing (Spracklen et al., 2023)

Limitations:

- High computational cost (up to 4 CPU days) and extensive manual effort

Traditional approaches, though effective, are resource-intensive and time-consuming, highlighting the need for more efficient, automated methods.

LLM-Based Streamliner Generation

New Approach:

- Use of Large Language Models (LLMs) to generate streamliners automatically

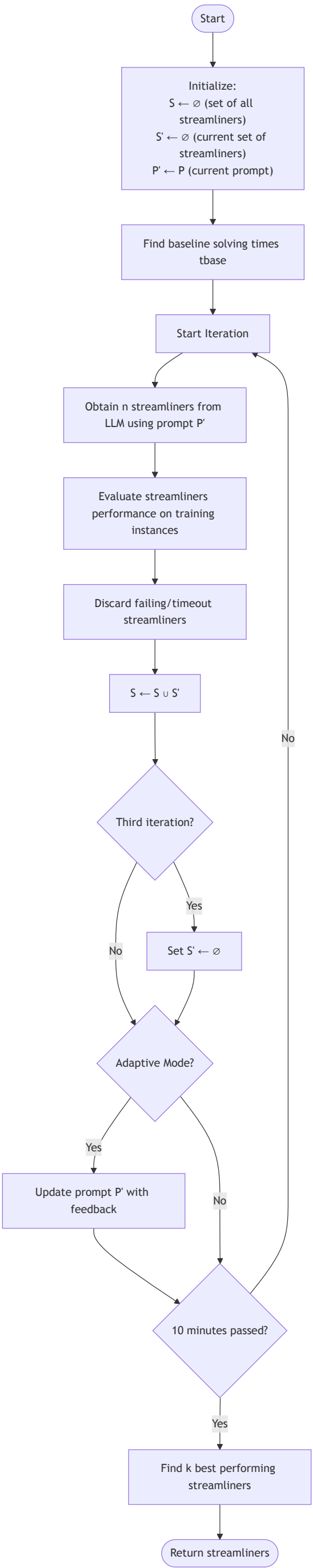
StreamLLM System:

- Combines engineered prompts, adaptive feedback, and quick empirical testing on MiniZinc models

Benefits:

- Real-time generation and potential for novel constraint combinations

The LLM-based approach integrates modern AI with empirical validation, significantly reducing the time needed to generate effective streamliners.



Realtime vs. Offline Generation Strat

Realtime Approach:

- Iterative prompting with testing on a small set of training instances (≈ 15 instances)
- Suited for hard problems where every minute saved is significant

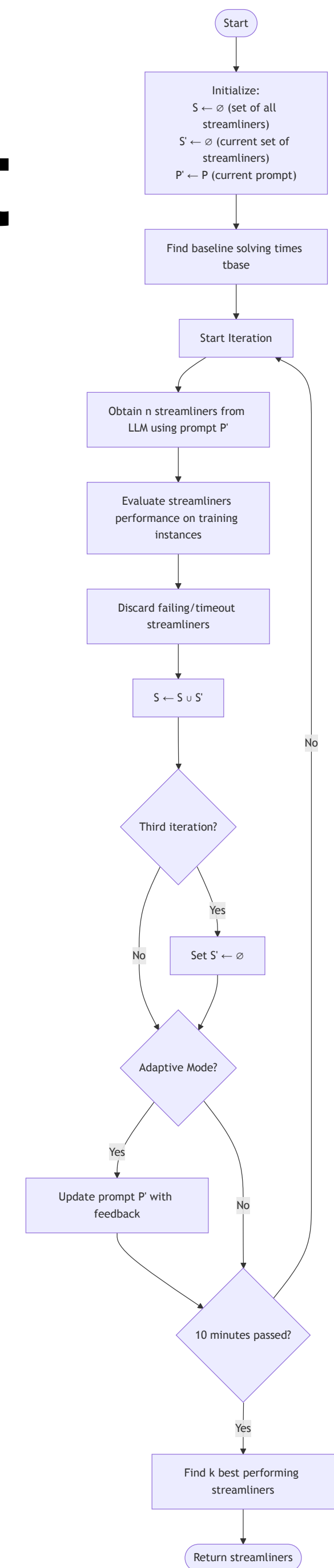
Offline Approach:

- Extended training over a larger instance set (up to 4 hours), yielding streamliners that save $>90\%$ solving time

Adaptive vs. Static:

- Adaptive methods integrate feedback, whereas static methods use fixed prompts

Comparing realtime and offline strategies reveals trade-offs between immediate results and long-term performance gains, with adaptive methods offering dynamic improvements based on test feedback.



Experimental Setup & Benchmark Problems

Models & Instances:

- MiniZinc models for problems like BIBD, Social Golfers, Hypergraph Coloring, etc.

Instance Generation:

- Custom difficulty calibration and evaluation across original, disguised, and obfuscated variants (Lee et al., 2023)

Setup

- We run the unstreamlined encoding and three streamlined encodings in parallel
- Utilises the complementarity of streamliners

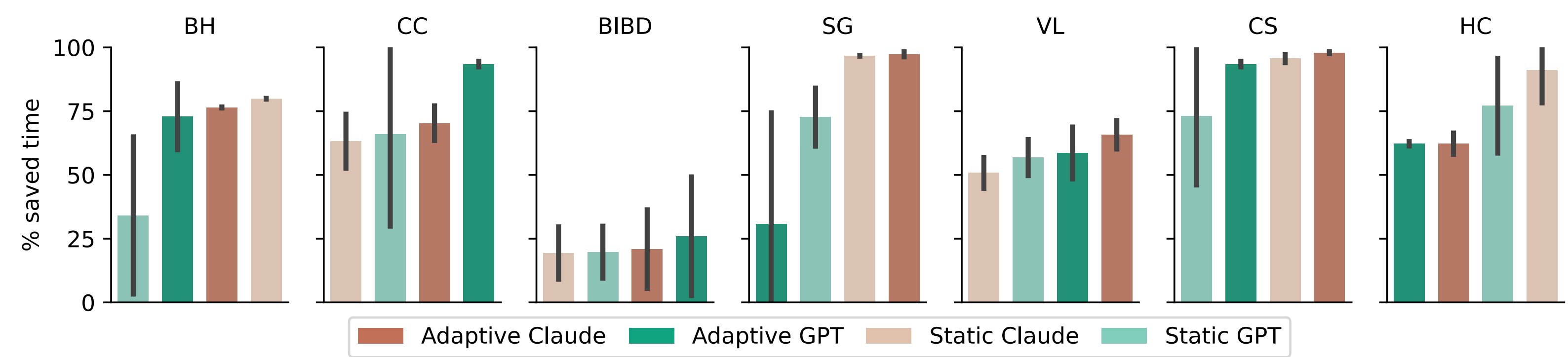
Metrics:

- Runtime reduction and robustness (unsat percentages)

The experimental setup ensures a comprehensive evaluation of the LLM-based approach across diverse problem variants, providing clear metrics for performance comparison.

1. BI Block Design (BIBD)
2. Car Sequencing (CS)
3. Social Golfers (SG)
4. Vessel Loading (VL)
5. Black Hole (BH)
6. Carpet Cutting (CC)

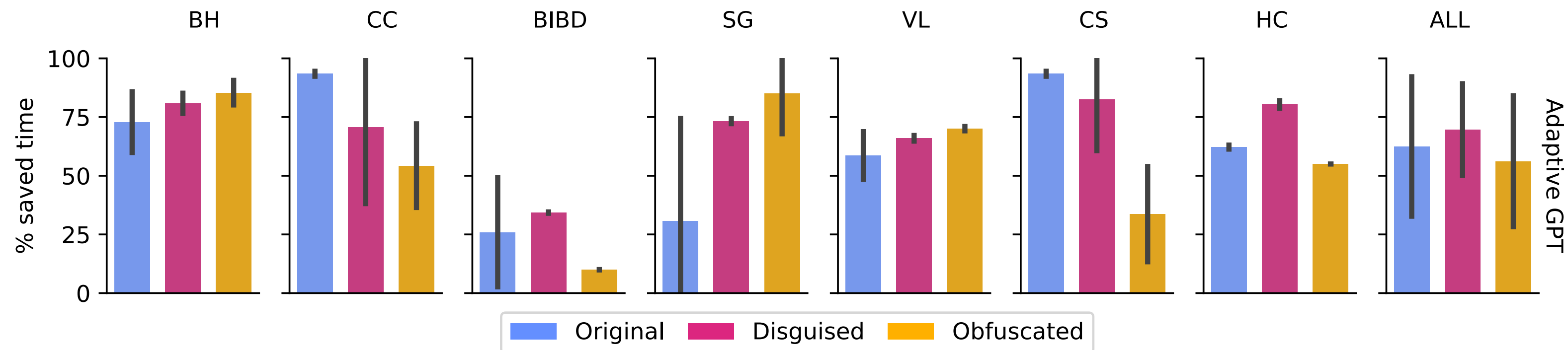
Key Results & Performance Improvements



Significant improvements in solving times are observed, with detailed comparisons underscoring the robustness and efficiency of the LLM-generated streamliners.

Obfuscation and Disguise

Disguised Social Golfer's problem. *The zookeepers at a large zoo are organizing seasonal habitat rotations to encourage socialization among animals. Each season, animals are grouped into mixed-species habitats of a fixed size. The zookeepers want to ensure that no two animals share a habitat more than once during the rotation schedule, while maximizing the number of season the rotations can last. This problem seeks for an allocation of animals to habitats which ensures maximum variety in animal pairings across seasons while maintaining the integrity of habitat sizes.*



The results underscore that using alternative, semantically rich identifiers (disguise) preserves—and can even enhance—the LLM's ability to generate effective streamliners. In contrast, stripping away these semantic cues through obfuscation significantly hampers performance, highlighting the critical role of context in understanding problem structure.

Offline Streamliner Generation Approach

Extended Training:

- Conducts training over longer durations (up to 4 hours) to explore a wider range of candidate streamliners.

Broader Instance Evaluation:

- Uses a larger set of training instances to assess streamliner performance more comprehensively.

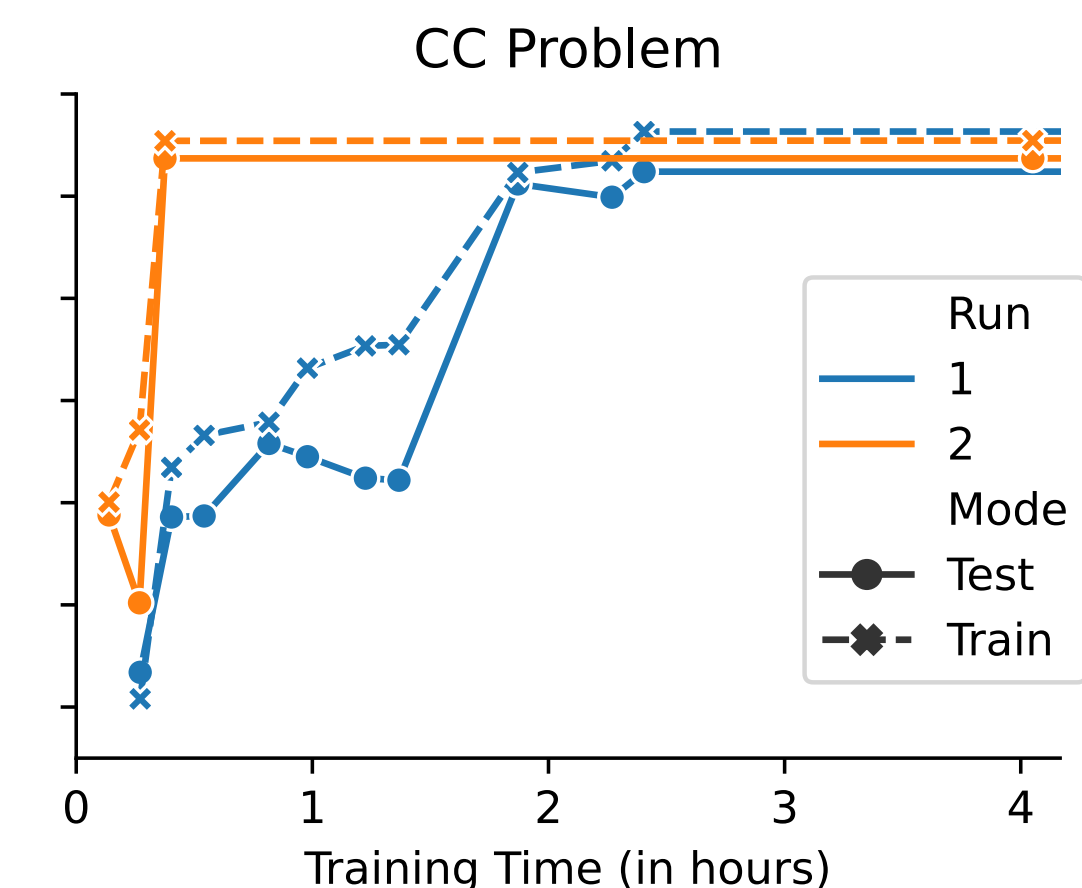
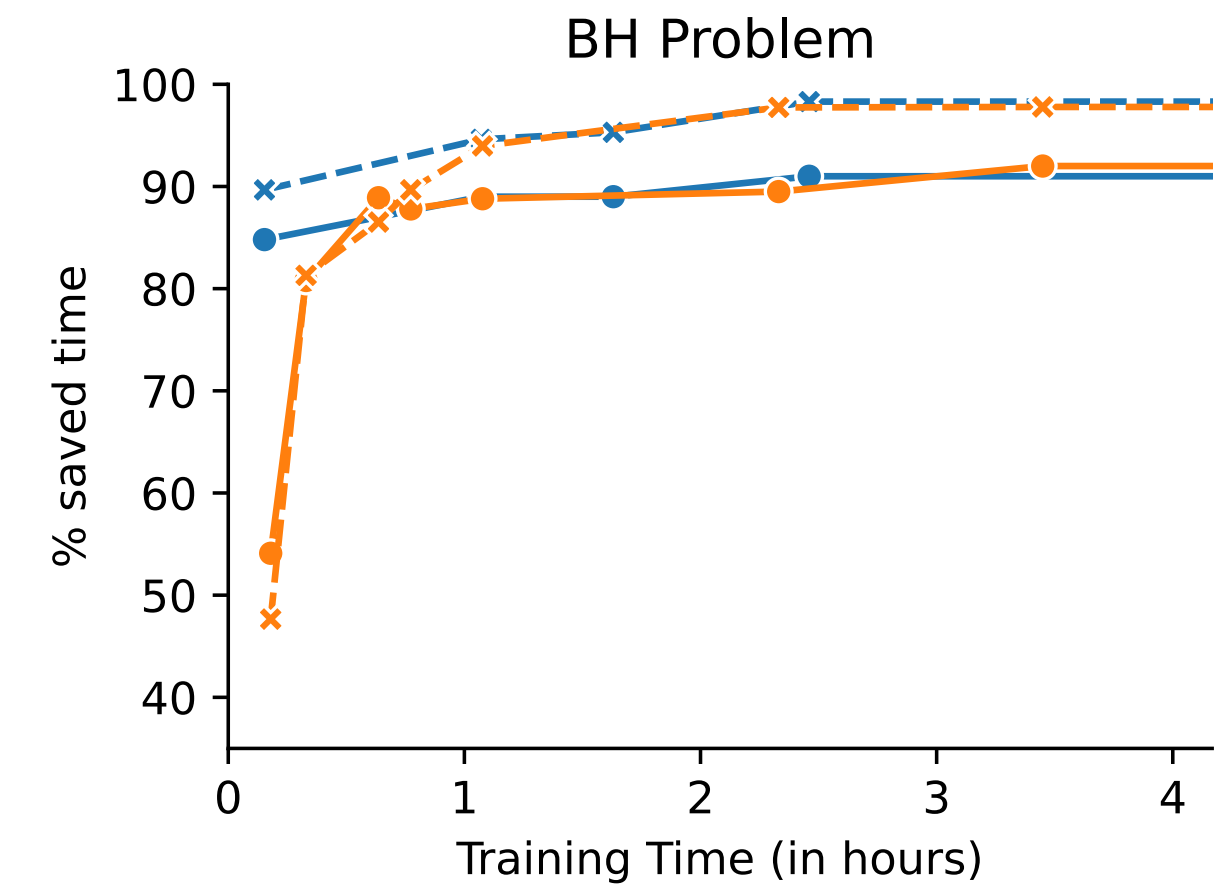
Enhanced Quality:

- Offline approach often produces streamliners that reduce solving times by over 90%.

Adaptive Feedback:

- Incorporates detailed feedback from extended runs, enabling refined prompt adjustments.

A detailed breakdown reveals the strengths and limitations of individual streamliners, emphasizing the value of combining multiple constraints for optimal performance.



Conclusions and Future Directions

Summary:

- LLM-generated streamliners significantly reduce solving times
- Both realtime and offline approaches are effective, with realtime showing practical gains

Future Work:

- Develop smaller, specialized LLMs through knowledge distillation
- Introduce formal verification for constraint soundness
- Explore meta-optimization via problem disguising

Technical Report at <https://arxiv.org/pdf/2408.10268>