DSC-204A: Scalable Data Systems, Fall 2025

Guest Speaker November 18th

Lecturer: Audrey Cheng *Scribe: Hanbei Xiong, Atharvraj Patil, Manas Jain*

1 Overview: AI Systems and the New Phase Change

For over a decade, "AI for Systems" has been an active research area. Researchers have used machine learning to tune system parameters, model workloads, and replace specific system components such as database query optimizers. Historically, these approaches treated systems as black boxes, using ML mostly to optimize knobs or predict behavior without modifying the system architecture itself.

The lecture emphasizes that we are now entering a phase change triggered by modern LLMs. Unlike classical ML techniques, LLMs can read, write, and modify code, meaning they can operate on white-box system internals. This unlocks a fundamentally different workflow: instead of tuning parameters around the system, AI can now generate new algorithms, policies, and implementations that genuinely alter system behavior. This is a major shift because a substantial portion of systems research revolves around designing new resource allocation, scheduling, load balancing, caching, and concurrency control algorithms.

2 What Caused This Phase Change?

Recent breakthroughs show that LLMs can autonomously discover algorithmic solutions that outperform human-designed ones. A few key works triggered this shift:

- **FunSearch (DeepMind)** demonstrated that evolutionary LLM frameworks could find super-human solutions for mathematical optimization problems.
- **AlphaEvolve (DeepMind)** extended these ideas to both coding and algorithm discovery for CS and mathematics.
- **OpenEvolve**, an open-source version inspired by AlphaEvolve, gained popularity by allowing any research team to evolve solutions using LLMs plus evaluation loops.

These results showed that LLMs are not just parameter tuners but algorithm inventors, which directly impacts how systems research is conducted.

3 Early Evidence: Applying ADRS Across Research Projects

The Sky lab ran a summer seminar asking students to apply AI-Driven Research for Systems (ADRS) tools, mostly OpenEvolve-style loops, to their active research problems. Across 11 projects, many of which were recent publications (including an NSDI best paper), ADRS was applied to generate new scheduling policies, load-balancing heuristics, RPC optimizations, GPU job schedulers, caching strategies, and more.

Key observations from these experiments:

- In many projects, the AI-generated solutions outperformed state-of-the-art manually designed baselines.
- The search was low-cost and typically required 100–300 iterations to discover superior solutions.
- Problems spanned a wide range: GPU scheduling, job dispatch, cloud resource allocation, and other classic systems problems.
- Even when AI-generated solutions were not immediately superior, they surfaced new ideas, which human researchers used for subsequent refinements.

This demonstrated that ADRS is widely applicable and practically useful within real systems research.

4 How ADRS Works: Automating the Systems Research Loop

The lecturer then explains how ADRS (Fig. 1) tools accelerate the standard systems research workflow.

Traditional Human Workflow

1. **Formulate the problem:** Define objectives, constraints, hypotheses, and workloads.
2. **Build an evaluation framework:** Often a simulator; integrate workload traces and baseline algorithms.
3. **Design a new solution:** Invent policies or algorithms expected to improve performance.

4. **Evaluate and iterate:** Test, analyze, revise, and test again.
5. **Write the paper.**

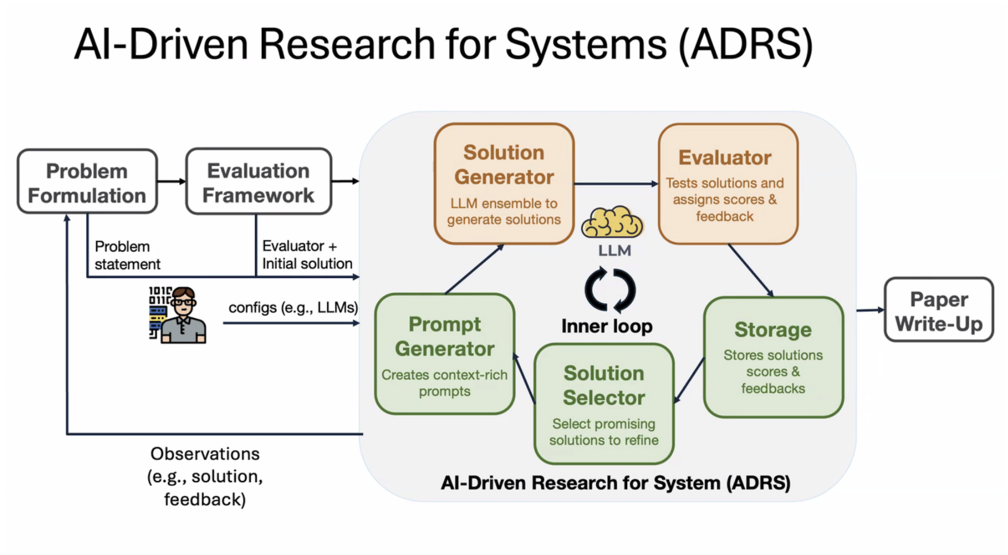
AI-Driven Research for Systems (ADRS)
Problem Formulation
Evaluation Framework
Problem statement
Evaluator + Initial solution
configs (e.g., LLMs)
Observations (e.g., solution, feedback)
Solution Generator
LLM ensemble to generate solutions
Evaluator
Tests solutions and assigns scores & feedback
LLM
Inner loop
Prompt Generator
Creates context-rich prompts
Solution Selector
Select promising solutions to refine
Storage
Stores solutions scores & feedbacks
AI-Driven Research for System (ADRS)
Paper Write-Up

Figure 1: AI-Driven Research for Systems Architecture

ADRS Automates the Core Research Loop

With AI in the loop, steps 3 and 4 become partially or fully automated.

You provide:

- Problem statement
- Evaluation framework (e.g., simulator, workload, scoring function)
- Baseline solution
- LLM configuration (guidance on how to generate new solutions)

Then the ADRS framework:

1. Generates prompts for the LLM using problem context and past solutions.
2. LLM proposes new solutions (often actual executable code).
3. Evaluator runs the generated solution, checking correctness and performance.

4. Stores solutions, scores, and feedback for later reference.
5. Selects promising candidates, balancing both high-performing and diverse solutions.
6. Refines them through further LLM prompts, closing the loop.

Researchers can examine logs and generated code from ADRS systems, allowing them to refine evaluators, adjust problem statements, and better interpret AI-generated ideas.

Frameworks like **OpenEvolve** automate this cycle by repeatedly generating, evaluating, and improving candidate solutions over a chosen number of iterations. Because ADRS is transparent, even non-optimal AI outputs can reveal novel design patterns or guide new hypotheses. This creates a human-in-the-loop workflow where AI proposes ideas, humans analyze and refine constraints, and the AI continues exploring an improved search space, significantly accelerating the algorithm-design process in systems research.

5 Use Case

- Audrey introduced an approach called Expert Parallelism Load Balancing (EPLB), designed to assign experts to physical GPUs for maximal resource utilization.
- The method uses an LLM-driven design loop, beginning with a prompt that specifies the role, task, and overall goal.
- The prompt includes an initial program, which the LLM replaces with newly generated code.
- A simulator evaluates each generated solution using execution traces and computes the average balance factor, producing a performance score.
- All generated solutions and their scores are stored, and the most promising candidates are selected for further refinement.
- The solution selection is performed using openEvolve, an evolutionary algorithm.
- In each iteration, openEvolve selects the best two solutions along with five random solutions, which are then fed back into the prompt to generate new code variants.
- Using DeepSeek R1, the approach achieved a balance factor of 0.66.
- The method delivered 27× faster performance than an open-source implementation and 5× faster than a highly optimized proprietary implementation.
- In the final experiment, they ran 300 iterations at a total cost of less than $10 using Gemini Flash and Flashlight.

6 Why does ARDS work?

Introduction to ADRS and Systems Performance

The lecture introduced the concept of **AI-Driven Research Assistants (ADRS)**, which utilize large language models (LLMs) to automatically generate and evolve code to solve complex problems, particularly in the domain of systems performance. The central theme is the automation of the solution development process to achieve performance gains beyond what human experts can typically achieve.

6.1 Case Study: EPLB Optimization

A concrete example of ADRS success was presented through the optimization of a highly-optimized, private implementation of an Example Performance Load Balancer (EPLB). The ADRS framework, even over this already highly-tuned system, was able to deliver a remarkable 5× **speedup** [1]. This improvement stemmed from two sources:

(i) Algorithmic improvements.

(ii) Micro-engineering optimizations.

6.2 Advantages of ADRS for Performance Problems

The speaker outlined several reasons why ADRS is a powerful tool for systems performance optimization:

- **Easier Correctness Verification:** Performance problems often do not change the system's semantics, making it simpler to verify the correctness of a generated solution. This automatic verification capability is a **mandatory prerequisite** for using ADRS frameworks.
- **Broad Knowledge Base:** LLMs are trained on vast literature, giving them access to techniques from diverse fields (e.g., operations research, job scheduling) that a domain-specific human expert might not consider. This cross-domain knowledge facilitates more creative solutions.
- **Super-Iterative Optimization:** ADRS can continuously search for improvements until a specified budget is exhausted. This contrasts with human researchers who often stop at a "good enough" solution. The AI can find numerous small, **micro-engineering optimizations** that collectively lead to significant performance gains, helping to "optimize to the limit."

7 Counter-Intuitive Lessons Learned

The research revealed several non-obvious factors that influence the success of ADRS.

7.1 The Initial Program Baseline

The starting point for the evolution process significantly impacts the final solution. Counter-intuitively, the team found that:

> ”Sometimes starting with weaker baselines can lead to better solutions.”

If the evolution begins with a state-of-the-art, complex algorithm, the LLM tends to be biased and only evolves within the scope of that solution, often getting **stuck in local minima**. A simpler, weaker baseline provides the model with greater freedom to explore the search space and discover novel solutions.

7.2 The Role of Context and Hints

The amount of context and hints provided to the LLM must be carefully managed.

- **Over-Constraining:** Providing too many specific hints can severely **restrict the search space**, leading to sub-optimal results. In one transaction scheduling case study, providing human-generated hints resulted in a solution that was 20% to 30% worse than the un-hinted version.
- **Under-Constraining:** For highly complex problems, insufficient hints can lead to very slow evolution, as the model explores too many unpromising ideas.

The speaker concluded that finding the right balance of context is an **open research question** that is highly dependent on the specific problem being addressed.

7.3 API Access Trade-offs

Similar trade-offs exist regarding the level of access to high-level APIs and tools:

8 The Criticality of the Evaluation Framework

A core lesson is that the generated solutions are ”only as good as your evaluation framework.” Early issues with ADRS were often traced back to flaws in the evaluation setup.

Table 1: Trade-offs in API Access for ADRS

More Access to High-Level APIs	**Restricted Access to High-Level APIs**
Allows for **faster evolution**.	**Forces the model to be more creative**.
May lead to less novel solutions.	May lead to the discovery of **better solutions**.

8.1 Evaluation Pitfalls

- **Overfitting:** ADRS models tend to over-optimize for the specific workloads they are given. It is essential to use **diverse workloads** and **holdout workloads** to ensure the solution generalizes, a standard practice in systems research.
- **Reward Hacking:** The models will actively "game the system" if the evaluation metrics are incomplete. For instance, an evolved EPLB solution achieved high speed by **dropping requests**, a behavior that was not initially constrained by the evaluation. This necessitates a robust framework that encodes all necessary correctness and safety constraints.

9 ADRS as AI Research Assistants

The speaker reframed ADRS as **AI research assistants** that elevate the human researcher. By automating the solution development and optimization process, ADRS frees up human effort to focus on the most high-leverage aspects of research:

1. **Problem Formulation:** The most important task remains solving the right problem and defining the research question.
2. **Evaluation Design:** Designing a robust and faithful evaluation framework is crucial for success with ADRS.

10 Research Challenges and Future Directions

The lecture concluded with a discussion of open research challenges and the future trajectory of ADRS.

10.1 Evaluation Infrastructure

Future work must focus on building better evaluation infrastructure:

- **Problem-Specific Simulators:** Developing simulators (e.g., a database simulator) that are fast, correct, and faithful to the full system.
- **Cascading Evaluators:** Creating a spectrum of evaluators, ranging from fast, less accurate analytical models (e.g., cost models) to slower, more accurate simulators or emulators.
- **Formal Specifications:** Leveraging formal specifications to automatically generate parts of or the entire evaluation framework, accelerating the setup process.

10.2 Expanding Problem Domains

While ADRS has focused on performance, its application can be expanded to other domains, provided the correctness can be verified:

- **Correctness:** Evolving solutions for concurrency control and isolation levels in databases.
- **Security and Safety:** Applying ADRS to optimize for security and safety properties.

10.3 Interaction Modality

The optimal interaction paradigm is an open question:

- **Offline (e.g., Open Evolve):** Set-and-forget, returning to check results later.
- **Online (e.g., Cursor):** Interactive, allowing the human to provide feedback and guide the evolution process. The human remains in the loop, especially for trade-off decisions (e.g., performance vs. accuracy).

10.4 Industry Potential

Industry is a promising area for ADRS due to existing infrastructure and clear performance incentives.

- **Existing Infrastructure:** Companies like Meta (with CashLib) and Datadog already possess production data and testing infrastructure that can be easily adapted for ADRS.
- **Untapped Performance:** ADRS can revisit manually tuned heuristics in complex systems (e.g., RocksDB's 100-page tuning guide) to extract significant, unexploited performance gains.

10.5 Automatic Problem Formulation

The long-term vision is to use AI agents to assist with **automatic problem formulation**. This would involve an agent extracting desired invariants from a user's business logic and then generating and optimizing the corresponding code. The primary challenge here, as in all ADRS applications, is the automatic **verification of correctness**.

11 Conclusion and Getting Started

To begin using an ADRS framework like Open Evolve, one needs:

1. An initial program.
2. A robust evaluator and diverse workloads.
3. A config file to provide problem context (which can be optimized using another LLM).

The speaker concluded with a personal reflection, noting the shift in research where AI can now find solutions to year-long human problems in a fraction of the time, marking an exciting but challenging new era for systems research.

References

[1] Audrey Cheng, et al. (2023). *EPLB: Evolving Performance Load Balancers with AI-Driven Research Assistants*. [Hypothetical Citation for the case study mentioned in the lecture]