Final Project

CS395T: Foundations of Machine Learning for Systems Researchers
Final Project

1 Objective

You can either pick one of these ideas, or suggest one of your own at a similar level of difficulty and novelty; submit a short proposal outlining your original idea and check with us ahead of submitting to have your idea approved.

Projects can be done individually or in groups of 2-3.

To discuss your idea, sign up for a 30 minute meeting to be scheduled during:

- October 2nd, Thursday 2-5pm
- October 7th, Tuesday 2-5pm

The signup link can be found here.

2 Ideas

2.1 ML/RL for video compression

Codec rate control is content-dependent and latency-constrained. Learning-based controllers can tune parameters to meet bitrate or quality targets better than fixed heuristics. How can we use a learning method to tune codec knobs or guide rate–distortion tradeoffs under bitrate/latency constraints? How do these methods compared to classical rate control baselines?

You can also look at other modalities for compression, such as image compression.

References to start with:

- MuZero for VP9 Video Compression Article on DeepMind's Website
- MuZero with Self-competition for Rate Control in VP9 Video Compression
- Deep Hierarchical Video Compression, Lu et al., 2023

Deliverables:

- Controller code
- Written report with performance results

2.2 Evolution for kernel code generation

Evolution can be used to discover solutions that humans otherwise might not have discovered. Can we use evolution to search for high-performance GPU/accelerator kernels? Where is the break-even point between search cost and performance benefits? The goal is to perform automated search and see if we can discover CUDA/Triton kernels that outperform compiler/library baselines without sacrificing correctness.

References to start with:

- DeepMind AlphaEvolve (open source implemenations include OpenEvolve)
- Towards Robust Agentic CUDA Kernel Benchmarking, Verification, and Optimization, Lange et al., 2025
- Ansor: Generating High-Performance Tensor Programs for Deep Learning
- CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning

Deliverables:

- Evolutionary algorithm or LLM+search pipeline for kernel generation
- Written report with results and examples of generated kernels

2.3 RL for decryption/decoding/deobfuscation

This project is recommended to be done in collaboration with Lain. Deobfuscation/decryption can be modeled as a sequential decision-making problem, where an agent must choose analysis or transformation actions step by step under partial observability to gradually recover hidden program semantics. The RL objective is to maximize progress toward uncovering correct code while minimizing cost.

References to start with:

- DOBF: A Deobfuscation Pre-Training Objective for Programming Languages, Roziere et al., 2021
- A Generic Approach to Automatic Deobfuscation of Executable Code

Deliverables:

- Custom environment or algorithm
- Written report

2.4 RL for self-verification in LLMs

Supervised training optimizes likelihood, not correctness; reliability improves when we optimize to pass verifiable checks (unit tests, parsers, proof checkers). The goal of this project is to look at tasks with ground-truth checkers (math proofs, code with unit tests, structured parsing) and train LLMs to generate solutions plus self-checks, rewarding only verifiably correct outputs.

References to start with:

- TinyZero (R1 Zero), a reproduction of the self-verifying DeepSeek model
- Reinforcement Learning with Verifiable Rewards Implicitly Incentivizes Correct Reasoning in Base LLMs
- Let's Verify Step By Step, Lightman et al., 2023

Deliverables:

- RL pipeline with verifiable rewards
- Written report with results compared to supervised baselines

2.5 RL for automated cyber defense

This project is recommended to be done in collaboration with Lain. Penetration testing involves planning multi-step exploits under uncertainty and defenses. Modeling the penetration testing problem as a POMDP, can we use a cyber gym to train penetration testing agents in a single agent or multi-agent RL setting? The goal is to use a simulator for learning cybersecurity policies.

References to start with:

- CybORG: An Autonomous Cyber Operations Research Gym, Baillie et al., 2021
- Microsoft CyberBattleSim
- Resources on RL for Cybersecurity
- Network Attack Simulator

Deliverables:

- Prototype pentest agent + evaluation scripts
- Written report

2.6 RL Sim Environment

This project is recommended to be done in collaboration with Lain. Many RL benchmarks are saturated or do not reflect current challenges in the field. Shareable benchmarks are always a huge bonus, and there are many open source game engines that can serve as the foundation of a gym style RL simulator (as in the case of VizDoom). Your goal is to port an open-source 3D engine into a gym style RL simulator to unlock a new benchmark for the RL community.

References to start with:

- VizDoom by Farama Foundation based on the ZDoom game engine
- OpenLara – open-source Tomb Raider game engine
- Gymnasium
- StableBaselines3 for baseline method implementations

Deliverables:

- Simulator (Gymnasium API, seeding, rendering, etc.) for an open-source game engine
- StableBaselines3 integration to test standard baselines
- Written report with baseline performance results

2.7 Adaptive speculative decoding for inference in LLMs

This project was suggested by Chris Rossbach. Speculative decode uses a small LLM to predict the output of a large LLM, which can be verified in linear time, so when it is right, it makes things go way faster. Like any speculative mechanism, if it is too frequently wrong, it harms performance. Can you train a model to predict whether speculation will work, and either turn it on/off or modulate the speculation depth? (This one will be a paper somewhere by someone eventually!).

References to start with:

- NVIDIA Blog post
- SpecDec++: Boosting Speculative Decoding via Adaptive Candidate Lengths, Huang et al., COLM 2025.
- BANDITSPEC: Adaptive Speculative Decoding via Bandit Algorithms, Hou et al. ICML 2025

Deliverables:

- Design documents and implementation of adaptive speculative decoding
- Integration into some open-source LLM
- Written report with performance results