The City College of New York

CSC 59866-E: Senior Project I

AI Agents for Decision Making in the Real World

By Saptarashmi Bandyopadhyay
Email: sbandyopadhyay@ccny.cuny.edu, sbandyopadhyay@gc.cuny.edu
Assistant Professor of Computer Science
City College of New York and Graduate Center at the City University of New York

March 25, 2026 CSC 59866

Agentic Systems Efficiency: Energy, Network bandwidth, memory, chip utilization and power optimization

Logistics and Motivation

Recall Lecture 16: We learned how to build and train Multi-Agent systems using PyTorch and PettingZoo.

The Reality Check: You've built a brilliant, mathematically sound multi-agent system. But when you deploy it, your cloud bill hits $10,000 in a day, your agents experience 5-second latency, and the GPU runs out of memory.

Today's Agenda

The Cost of Agency: Energy, Bandwidth, and Memory bottlenecks.

Memory Math: Understanding the KV Cache.

Compute & Power Optimization: Quantization and Mixture of Experts (MoE).

AlphaChip: How Deep RL is designing the next generation of AI hardware.

The Cost of Agentic Systems

Why Efficiency Matters in AI Agents

Standard LLMs vs. Agents: A human chatting with ChatGPT generates a few hundred tokens. An autonomous LLM AI agent "thinking" in a loop (Plan -> Execute -> Observe -> Re-plan) generates *tens of thousands* of tokens per minute.

The Four Bottlenecks of Scale:

- **Compute/Energy:** FLOPs required for matrix multiplications drain power.
- **Memory:** Storing context histories (KV Cache) rapidly exhausts GPU VRAM.
- **Network Bandwidth:** Moving data between decentralized agents or fetching from the cloud causes massive latency.
- **Chip Utilization:** Inefficient code leaves expensive silicon sitting idle waiting for memory.

Optimizing Network Bandwidth

Multi-agent systems (like swarms of drones) cannot always rely on Cloud APIs due to bandwidth constraints and latency.

Semantic Routing: Instead of sending raw, high-resolution images to the cloud, Edge Agents use tiny, local visual models to extract semantic meaning (e.g., "Car at coordinates X,Y") and only transmit a few bytes of text.

Token Pruning / Summarization: Agents must summarize their own scratchpads. If Agent A sends its thought process to Agent B, the prompt size linearly increases network payload. *Compression is key to decentralized agency.*

Memory Bottlenecks

To generate the 1,000th token, an LLM agent must look at the previous 999 tokens. Recalculating the attention for all 999 tokens every single step wastes enormous compute.

The Solution (KV Cache): We cache the Key (K) and Value (V) tensors of past tokens in GPU Memory (VRAM). Compute is saved, but VRAM consumption skyrockets.

The Agentic Implication: As your agent loops and its context window grows, the KV Cache will eventually cause an Out-Of-Memory (OOM) error, killing the agent.

The KV Cache Equation

How much memory does an agent actually consume? It is entirely deterministic.

The VRAM (in bytes) required to store the KV cache for a single token is:

$$\text{Memory per Token} = 2 \times n_{\text{layers}} \times d_{\text{hidden}} \times p_{\text{bytes}}$$

- 2 : Because we store both Keys and Values.
- n_{layers} : The number of layers in the Transformer.
- d_{hidden} : The hidden dimension size of the model.
- p_{bytes} : Bytes per parameter (e.g., 2 bytes for FP16).

Total VRAM:

$$\text{Total Memory} = \text{Memory per Token} \times \text{Sequence Length} \times \text{Batch Size}$$

Compute, Chip Utilization, & Power Optimization

Optimizing the Math: Quantization

To improve chip utilization and power efficiency, we alter the math itself.

Recall: Quantization converts neural network weights and activations from high-precision (32-bit floating point, FP32) to lower precision (8-bit or 4-bit integers, INT8/INT4).

- **Pros:** Drastically reduces Memory bandwidth requirements (which consumes the most power) and increases matrix multiplication speed.
- **Cons:** Introduces minor rounding errors. Modern algorithms mathematically calibrate the network to preserve the agent's reasoning capabilities despite the lower precision.

Optimizing the Architecture: Mixture of Experts (MoE)

The FLOPs Bottleneck: Passing a token through a dense 70B parameter model uses massive amounts of electrical power.

Mixture of Experts: We break the dense network into smaller "Experts".

The Mechanism: A router network dynamically selects only top-K experts for each specific token.

The Efficiency Win: A 47B parameter MoE model might only use 13B active parameters during inference. This provides the intelligence of a massive model with the energy cost of a small one!

AlphaChip - AI Designing Better AI Hardware

Hardware Bottlenecks

We've optimized the software. But to run the next generation of agents, we need more efficient physical silicon chips (like Google's TPUs).

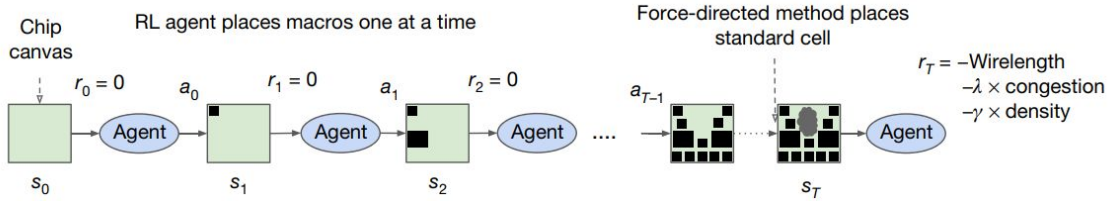
Chip Floorplanning: The process of placing thousands of macroscopic building blocks (SRAM memory macros) and millions of standard logic cells on a tiny silicon die.

The Problem: A poorly arranged chip requires longer wires, which drastically increases power consumption, heat, and processing latency. For decades, this took teams of human engineers *months* to solve via heuristics.

Chip Design as RL

In 2021, Google researchers formulated chip floorplanning as a Reinforcement Learning problem, later named **AlphaChip**.

- **The State:** An Edge-Based Graph Convolutional Neural Network (GCN) processes the netlist (the graph of how components connect) to understand the current layout.
- **The Action Space:** Placing the current macro sequentially on a discrete grid.
- **The Outcome:** The RL agent generates superhuman chip layouts in under 6 hours, vastly outperforming months of human effort.



The AlphaChip Reward

How do we define a "good" chip for the RL agent?

The RL agent optimizes a proxy cost function matching real-world physics:

$$J(\pi) = \mathbb{E}_{\pi}\left[-\left(\text{Wirelength} + \lambda \cdot \text{Congestion} + \gamma \cdot \text{Density}\right)\right]$$

Wirelength: Sum of half-perimeter bounding boxes of all wire nets. (Shorter wires = faster chips & less power).

Congestion: Prevents placing too many interconnected blocks in the same corner, which would make physical wire routing impossible.

Density: Ensures components are evenly distributed across the silicon.

Transfer Learning in Hardware Design

Previous heuristic tools started from scratch every time. AlphaChip *learns* the underlying physics of chip design.

By pre-training the RL agent on a diverse set of older chips, it learns universal representations of routing and congestion.

When tasked with designing a brand-new TPU architecture, the pre-trained agent converges to a superhuman layout exponentially faster than an agent starting from scratch.

The Bitter Lesson

The 2024 Addendum: The AlphaChip team recently highlighted how this work embodies Richard Sutton's "Bitter Lesson" in AI.

The Lesson: Human researchers constantly try to build their domain knowledge into systems (heuristics). But in the long run, general methods that leverage massive computation and learning (like Deep RL) *always* win.

Real-world Deployment: AlphaChip is not just a research paper. It has been used to physically design Google's TPU v5e and TPU v6 Trillium chips!

Questions?