AI-Generated Cache Replacement Policies with Groq API

Amna Hassan
Department of Computer Science, UET Taxila
Email: amnahassan.ahf@gmail.com

September 18, 2025

1 Introduction

Caches are critical for bridging the speed gap between CPU and main memory. Their effectiveness depends on the cache replacement policy, which decides which line to evict on a miss. Traditional approaches such as LRU and SHiP work well in some workloads but are far from optimal across the board.

This project explores whether Large Language Models (LLMs) can generate novel cache replacement policies automatically. Using Groq API, I prompted a variety of state-of-the-art open models to produce candidate policies. The generated code was integrated into ChampSim CRC2, compiled, and evaluated.

Although I am approaching this topic as a beginner, the work demonstrates that self-motivated exploration with AI tools can yield competitive policies, highlighting potential for AI-assisted hardware research.

2 Background

Cache replacement policies attempt to approximate Belady's optimal policy, which evicts the block whose reuse is farthest in the future. Since this is not implementable directly, heuristics are employed:

- **LRU:** Evicts least recently used block.
- **SHiP:** Predicts reuse behavior using signatures.
- **Hawkeye:** Trains predictors to imitate Belady's decisions.

These methods are hand-designed. The central question here is: can generative AI models propose competitive strategies without human intervention?

3 Design

The workflow consisted of:

1. Prompting LLMs with known policy templates.
2. Parsing and compiling generated code in ChampSim CRC2.
3. Running five workloads: *astar, lbm, mcf, milc, omnetpp.*
4. Collecting IPC and other statistics.
5. Comparing performance against baseline policies.

4 Methodology

- **Simulator:** ChampSim CRC2.
- **Benchmarks:** SPEC traces (astar, lbm, mcf, milc, omnetpp).
- **Metrics:** IPC (instructions per cycle), miss statistics, metadata overhead.
- **Models tested:** GPT-OSS-20B/120B, Moonshot Kimi-K2 (and 0905), Qwen-3 32B, DeepSeek R1 Distill 70B, Meta-LLaMA Maverick/Scout, Gemma-2 9B.
- **Iterations:** 5 generations per model.

5 Results

5.1 Cache Hit Rates by AI Model

The main contribution of this project is systematically running many LLMs and documenting results. Table 1 shows average cache hit rates for all generated policies.

Table 1: Average Cache Hit Rates of Policies Generated by Different AI Models

Model	**Policy**	**astar**	**lbm**	**mcf**	**milc**	**omnetpp**	**Avg Hit**
GPT-OSS-20B	ASeR	21.00	12.34	12.33	7.25	85.48	27.68
	SRWR	41.18	37.80	33.44	27.06	34.90	34.88
	RDPR	25.59	12.17	34.20	6.11	84.67	32.55
	CAAR	45.59	43.52	41.29	31.11	69.02	**46.11**
GPT-OSS-120B	EGAA	45.57	43.99	41.66	32.20	63.10	**45.30**
	DRD-FA	41.43	37.06	33.33	26.70	30.61	33.83
Kimi-K2-0905	VAH	39.76	22.22	48.74	11.71	44.54	33.40
	T-MAP	44.96	43.99	37.48	32.19	48.61	**41.45**
Kimi-K2	ChronoEntropy	29.19	15.45	40.96	6.43	85.46	**35.50**
	LunarSieve	28.82	16.54	27.93	10.24	83.07	33.32
Qwen-3 32B	SAPO	37.37	30.23	24.16	16.07	43.06	30.18
	DAC	36.13	25.48	47.25	9.68	78.04	**39.32**
	TIAR	0.81	1.89	1.03	0.89	0.07	0.94
DeepSeek-70B	SCAR	4.51	5.88	21.43	0.95	0.12	6.58
Meta-LLaMA Maverick	NeuCache	0.81	1.89	1.03	0.89	0.07	0.94
	FRACTAL	0.81	1.89	1.03	0.89	0.07	0.94
Meta-LLaMA Scout	CGBCR	0.81	1.89	1.03	0.89	0.07	0.94
	HTMS	0.81	1.89	1.03	0.89	0.07	0.94
Gemma-2 9B	DBP Cache	0.81	1.89	1.03	0.89	0.07	0.94
LLaMA-3.1 8B Instant	*Compilation Failed*	-	-	-	-	-	-
LLaMA-3.3 70B Versatile	*Compilation Failed*	-	-	-	-	-	-

5.2 Best Policy Per Model

Table 2 shows the best performing policy from each AI model tested.

Table 2: Best Performing Policy Per Model

Model	Best Policy	Avg Hit Rate
GPT-OSS-20B	CAAR	46.11
GPT-OSS-120B	EGAA	45.30
Kimi-K2-0905	T-MAP	41.45
Kimi-K2	ChronoEntropy	35.50
Qwen-3 32B	DAC	39.32
DeepSeek-70B	SCAR	6.58
Meta-LLaMA (Maverick/Scout)	FRACTAL / HTMS	0.94
Gemma-2 9B	DBP Cache	0.94

5.3 Parse and Compile Success Rates

Table 3 shows the compilation success rate for each model, indicating code quality and correctness.

Table 3: Compile/Parse Success Rate by Model

Model	Attempted	Success	Failed	Rate (%)
GPT-OSS-20B	5	4	1	80
GPT-OSS-120B	5	2	3	40
Kimi-K2-0905	5	2	3	40
Kimi-K2	5	2	3	40
Qwen-3 32B	5	2	3	40
DeepSeek-70B	5	1	4	20
Meta-LLaMA Maverick	5	2	3	40
Meta-LLaMA Scout	5	2	3	40
Gemma-2 9B	5	1	4	20
LLaMA-3.1 / 3.3	5	0	5	0

5.4 IPC Results and Rankings

Table 4 summarizes IPC values for key policies, while Table 5 ranks them by average IPC.

Table 4: IPC Results of AI-Generated Policies on 5 Workloads

Policy	astar	lbm	mcf	milc	omnetpp	Avg IPC
CAAR	0.1045	0.5329	0.0706	0.3636	0.4539	0.305
ChronoEntropy	0.1152	0.5948	0.0719	0.4013	0.4684	**0.330**
EGAA	0.1041	0.5360	0.0708	0.3614	0.4484	0.304
DAC	0.1128	0.5447	0.0768	0.3728	0.4617	0.314
T-MAP	0.1039	0.5368	0.0682	0.3631	0.4401	0.302

Table 5: Ranking of Policies by Average IPC

Policy	**Avg IPC**
ChronoEntropy	**0.330**
DAC	0.314
CAAR	0.305
EGAA	0.304
T-MAP	0.302

5.5 Area Analysis

Table 6 shows metadata overhead for each policy, based on cache configuration (2 MB, 64B block, 16-way associativity, 32,768 blocks).

Table 6: Area Overheads of AI-Generated Policies

Policy	**Total Metadata**	**Area (KB)**
EGAA	32,768 $\times$ 18 bytes	576
ChronoEntropy	32,768 $\times$ 10 bytes	320
DAC	32,768 $\times$ 4 bytes	128
T-MAP	32,768 $\times$ 1 + extras	48
CAAR	32,768 $\times$ 1.5 bytes	48

Policies like T-MAP and CAAR fit within the 64 KiB budget, while EGAA and ChronoEntropy exceed it unless optimized.

6 Best Policy: ChronoEntropy

ChronoEntropy combines **temporal recency** with **entropy-based predictability**.

- Lines recently used are favored.
- Lines with stable, predictable reuse patterns are prioritized.
- Random/noisy lines are evicted.

Its advantage lies in balancing short-term recency with long-term predictability, making it robust across workloads.

7 Conclusion

This project demonstrates that LLMs can autonomously propose hardware-level cache policies. By systematically testing multiple AI models, I found that ChronoEntropy provided the best IPC (0.33).

The work also highlighted practical considerations: compilation failures, metadata overhead, and trade-offs between accuracy and feasibility. This experience reflects my ability to self-learn and explore new research directions, which I aim to extend as a motivated PhD student.

References

[1] ChampSim CRC2, https://github.com/ChampSim/ChampSim

[2] Groq API, https://groq.com

[3] GroqCloud Platform, https://console.groq.com

[4] A. Jain and C. Lin, "Back to the Future: Leveraging Belady's Algorithm for Improved Cache Replacement," ISCA 2016.

[5] Meta AI LLaMA Models, https://github.com/facebookresearch/llama

[6] Meta LLaMA 3 Series, https://huggingface.co/collections/meta-llama

[7] Qwen Large Language Models, https://github.com/QwenLM/Qwen

[8] DeepSeek Models, https://github.com/deepseek-ai/DeepSeek-LLM

[9] Google Gemma Models, https://github.com/google/gemma_pytorch

[10] Moonshot AI Kimi Models, https://platform.moonshot.cn

[11] GPT-OSS Open Source Models, https://huggingface.co/microsoft/DialoGPT-large

[12] Hugging Face Model Hub, https://huggingface.co/models

[13] SPEC CPU Benchmarks, https://www.spec.org/cpu/