Vrije Universiteit Amsterdam

THESIS DEFENSE

Presented by M’hamed Belalia

ABSTRACT

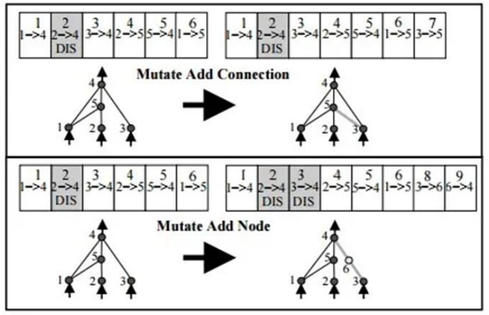
Neuroevolution algorithms face computational bottlenecks with 70%+ of time spent on repetitive neural network evaluations. This research implements generational caching using LRU eviction to store neural computation results across NEAT generations.

OVERVIEW

INTRODUCTION

- Traditional ML: fixed architectures trained with gradient descent
- Requires Labeled Data and differentiable functions
- Limited to predefined Network structures

- NeuroEvolution : evolves both network topology and weight simultaneously
- No need for labeled data or differentiable functions
- Neat algorithm: starts minimal, grows complexity naturally
- Proven success in games, robotics and control systems



PROBLEM

1

Computational bottleneck 70%+ of time spent on neural network evaluations

2

Massive scale : Millions of neural computations per experiment (500 * 50 generation * thousands of evaluations

3

Repetitive Waste Identical neural computations repeated across generations

THEORETICAL

Cache

- Temporal Localitity : Recently accessed data likely to be accessed again
- Spacial locality: similar computations cluster together
- LRU Principle Least Recently Used items are least likely to be needed

Neuroevolution caching

- Generational Similarity: Networks share common substructures across generations
- Computation Reuse: Identical/similar neurons calculations repeat frequently
- Progressive Learning : Cache effectiveness improves over evolutionary time

Insight

- Precision vs Diversity Trade-off: too precise = few hits, too coarse = reduced evolution
- Cache size optimization: Balance between memory and effectiveness
- Evolutionary Preservation: Caching must not compromise genetic diversity

OBJECTIVES

Objective 1

Design and Implement Generational Caching System

Develop a persistent caching mechanism that stores neural computation results across NEAT generations using LRU eviction strategy. The system must cache individual neuron computations rather than just network outputs, with precision-controlled cache keys to balance hit rates with genetic diversity preservation.

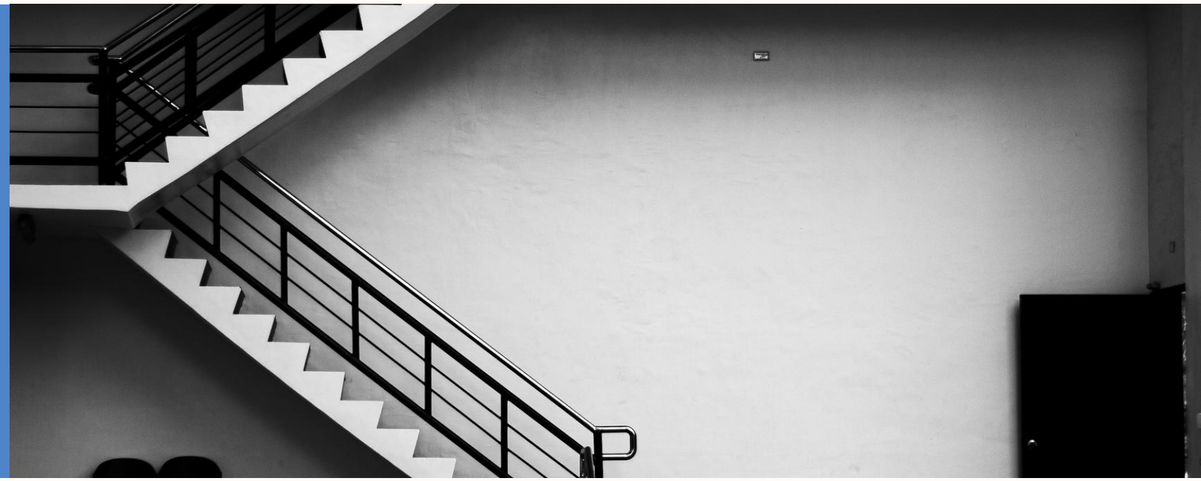
Objective 2

Validate Performance Improvement While Maintaining Solution Quality

Experimentally demonstrate computational speedup through controlled comparison on Atari Breakout benchmark. Measure cache hit rates, computation time reduction, and memory overhead while ensuring no degradation in evolutionary effectiveness or final solution fitness scores.

HYPOTHESIS

Implementing generational caching with LRU eviction strategy can accelerate neuroevolution by 5-10% without compromising solution quality. The cache will demonstrate progressive learning across generations, achieving hit rates of 20-30% while maintaining genetic diversity through precision-controlled quantization. Memory overhead will remain negligible (~3-5MB) compared to computational savings, making this approach practically viable for neuroevolution research.



Design

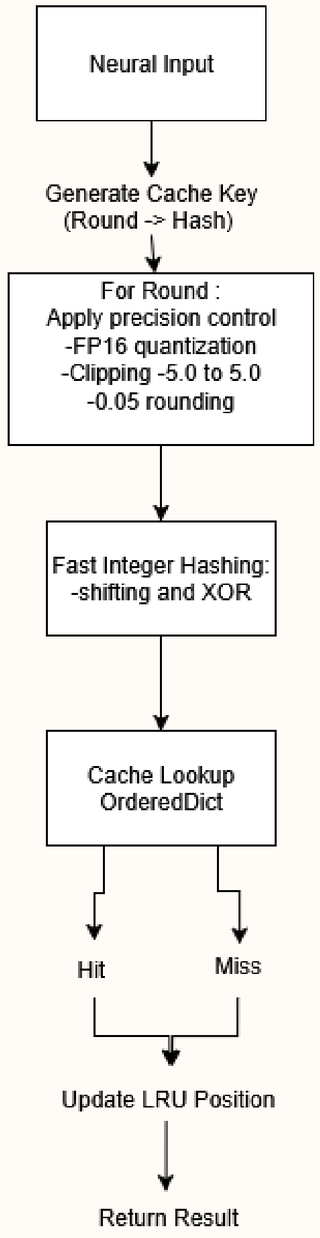
Architecture

Generational Caching:

- OrderedDict-based LRU cache with 50k to 150k
- Persistent storage across NEAT generations
- Individual neuron computations caching(not just network output)
- Fast Interger Hashing for O(1) cache operation

Cache Management:

- Growth phase: Cache expands
- Trimming phase: LRU eviction between generations to target size
- Precision control: fp16 quantization with -5 to 5 clipping and 0.05 resolution
- Conditional caching: Skip computations unlikely to benefit



Design

Overall design

Neat Parameters:

- Population: 500 genomes per generation
- Activation Functions: multi-component (tanh, sigmoid, swish)
- Mutation rates: Standard Neat defaults
- Fitness function: Score-Based with behavioral incentives

Methodology:

- Mutilple cache size(50k,100k,150k) for optimization
- 50 generations per experiment for statistical significance
- Same parameters across all configuration

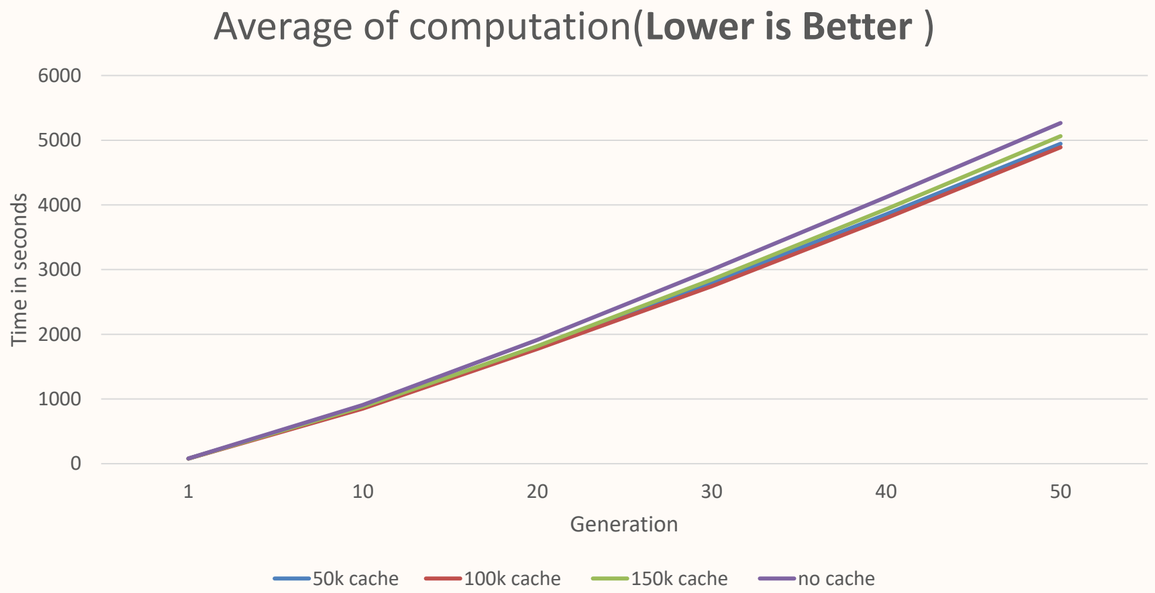
Controled Comparaison:

- Baseline: Standard NEAT without caching
- Treatment: NEAT with generational caching
- Environement: Atari Breakout
- Metrics: Computation time, cache performance, fitness etc...

Experimental Scale:

- Total evaluations: 25,000 network assessments per experiment
- Neural computations: 25+ million per experiment
- Baseline duration: ~87 minutes (50 generations)

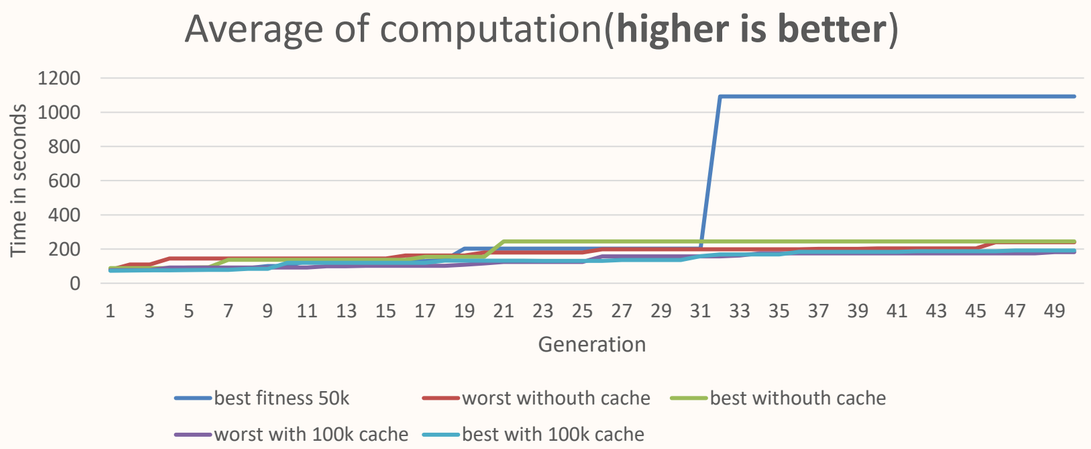
RESULT

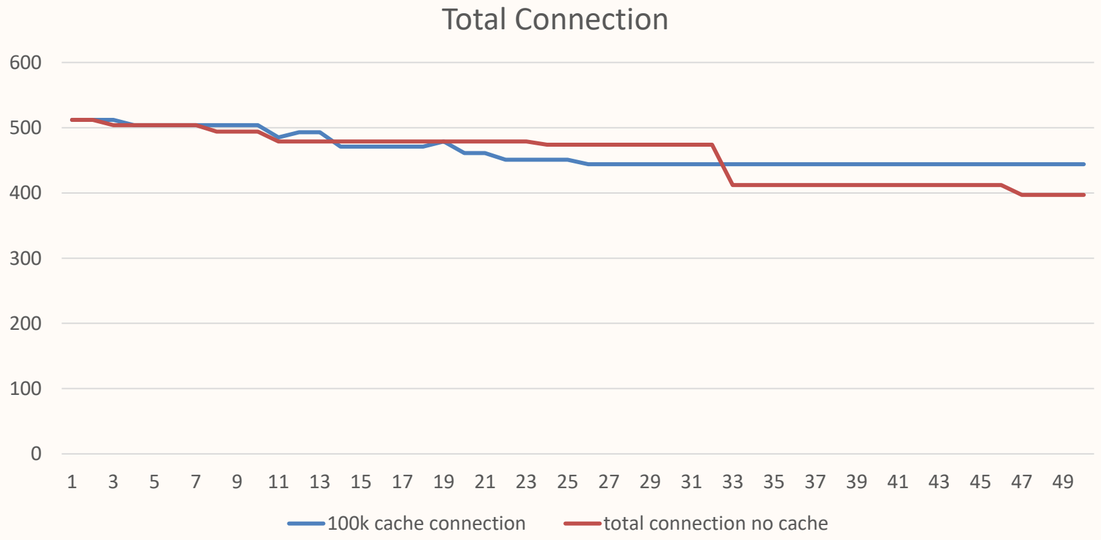


Generation:	50k cache	100k cache	150k cache	No cache
1	79.246	78.483	78.65	80.283
10	881.649	851.423	882.12	906.991
20	1805.52	1775.133	1816.095	1914.248
30	2803.058	2743.003	2846.935	2996.947
40	3854.866	3793.302	3932.984	4118.136
50	4945.414	4890.929	5063.021	5265.959

Configuration	Total Time (50 gens)	Time Saved	Speedup	Improvement
100K Cache	4890.929s (81.5 min)	375.0s	1.077×	**7.1% faster**
50K Cache	4945.414s (82.4 min)	320.5s	1.065×	6.1% faster
No Cache	5265.959s (87.8 min)	0s	1.0×	0% (baseline)

RESULT





Generation:	Best 50K	Worst with no cache	Best with no cache	Worst with 100k	Best with 100k
50	1092.6034	240.039	244.7916	182.2436	191.8926

Metric	100K Cache	No Cache	Difference
Total Nodes	6	5	+1 (+20%)
Total Connections	444	397	+47 (+12%)
Hidden Nodes	2	1	+1 (+100%)
Network Layers	3	2	+1 (+50%)

Generation	100k cache total connection	total connection no cache
50	**444**	397

Insight:

- High variability typical of evolutionary algorithm across all configurations
- 50k cache breakthrough: Achieved high performance with fitness of 1092.6 showing cache doesn't limit potential
- Consistent performance: most runs converged to similar fitness ranges (180-250)
- No systematic degradation: Caching doesn't change much solution quality
- Network complexity preserved :Caching enabled evolution of more sophisticated architectures (6 nodes, 3 layers vs 5 nodes, 2 layers) with 12% more connections, demonstrating that speedup doesn't compromise structural exploration

CONCLUSION

Main Findings:

- **Hypothesis Validated:** Achieved 7.1% performance improvement, falling within predicted 5-10% range
- **Speed Without Sacrifice:** Reduced computation time from 87.8 to 81.5 minutes while preserving solution quality
- **Enhanced Evolution:** Caching enabled more complex network architectures (6 nodes, 444 connections vs 5 nodes, 397 connections)

RECOMMENDATION

1

Develop Better caching system:

Implement faster hash functions and fuzzy matching within tolerance ranges to reduce lookup time and cache misses, while adding predictive prefetching based on network evolution patterns. These optimizations could potentially increase the current 7.1% speedup to 10-15% performance improvement.

2

Expand Research Scope:

Apply generational caching to different NEAT applications beyond Atari games, testing on robotics , real-time strategy games, and neural architecture search problems with longer experiments (100+ generations) to validate performance across diverse domains. Additionally, conduct multiple independent runs with statistical analysis and test with varying population sizes (100-500 genomes) to establish more robust performance baselines.

THANK YOU

Presented by M’hamed Belalia
Vrije Universiteit Amsterdam

https://youtu.be/9n97wr2o7gc