Slide 1:

Good Afternoon. Im M’hamed Belalia and today I’ll be presenting my bachelor thesis on ‘Accelerating Neuroevolution through generational Caching”

This thesis addresses a bottleneck in evolutionary algorithm and we propose a novel caching solution that achieves significant performance improvement while maintinaining solution quality

Slide 2:

Let me star with the core problem> Neuroevolutiuon algorithms face some computation bottleneck with over 70% of execution time cosumed by repertitive neural network evalution. This is a massive waste of computational ressources.

My thesis tacles this inefficiency by implementing a generation caching system using LRU eviction strategies. The key ionnnovatiobn is storing and reusing neural computation result across neat rather than recalculating identical operations. Because I neuroevolution the next genereation take som e characteristic of last generation it’s the crossover of some genome

The results speak for itselves: we achieved a 7.1% performance improvement on Atari Breakout while maintaining complete solution quality. This may seem modest, but it represent a large time saving that compound across longer experience.

Slide 3:

Todays presentations will follow this structure I will introduce you to neuroevolution then I will established the problem context, eexplain our theoretivcal approach, present our objectives and methodology ans show expereimental results, and conclude with implications and future work

Slide 4:

To understand why this research matters, we need to compare 2 paradigms in machine learning.

Traditional ML relies on fixed architectures trained with gradient descent. This approach requires labeled data , differientiable functions and predefined network structurews. While powerful it’s limited to problems where thoses constraints can be satisfied.

Neuroevolution takes fundamentally different approach it evolves both network topology and weight simultaneously, requiring no labeled data or differentiables functions. The Neat algorithm, which we foxcus on starts with minimal networks and grows complexity naturally trough evolution.

Neat has proven itself successful across diverse domains from game AI like Atari games, to robotics control systems, to complex optimization problems. Its ability to discover both architecture and parameters makes it better for domains where traditional ML struggles.

However, this power comes with a computational cost that grows as it scale.

Slide 5 :

The problem we are addressing has three critical dimension.

First, there is a massive computational bottleneck. Some analyses reveal that iover 70% of total computation is spent on neural network evaluations during the genome assement. This is a big inefficiency and it grows the more its scale like we can see the on the second problem. A typical expereiment with 500 genomes over 50 generation requires millions of neural computation. We are talking about 500 times 50 generation times thousands of of evaluation per genome. That a lot of calculation done. Lastly we have the most important one , the repetitive waste due to the incremental nature of evolutionary mutations, identical computations are repeated across generations. When a genome undergoes a minor mutation like adding a single connection or slightly adjusting the weight the majority of the neural computations remain identical to the parent genome. Yet current implementations recomputes every activation from scratch.

This represent a failure to exploit heritage of the last generation thus losing time and computation overtime.

Slide 6 :

Our solution is build on Three theoretical pillar

FDirst classical caching principles. We leverages temporal locality recently used data is likely to be accessed again, we use special locality similar computations are together, and we apply the LRU principle, leat recently used items are least likely to be needed thus we discard them

Second we have the insight on the neuroevolution. We observe generational similarities, some networks share common substructures across generation. This enable the computatiuon reuse, where identical neural calculation that are repeated frequently aqre reused, Most importantly, we see progressive learning overtime because the cache improves overtimes and cache most common used computation.

Third, We must balance precision versus diversity, too precise caching will yields to few hits, too coarse will lead to a reduces diversity and evolutionary effectivenees. Cache size optimization requires blancing memory and effectiveness because as the cache grow the lookup time grows also . And lastly caching must not compromise genetic diversity. This principle will guide how we implement our design.

Slides 7 :

This thesis had 2 primary objectives

The first one is to design and implement generational caching that caching mechanism stores neural computations result across generation using lru eviction strategy, the system must cache individual computations rather than just network output , with precision cache keys to balkance hit rates and genetic diversity

Secondly we need to validate the perofomance improvement whiule maintaining solutions quality. Thus we will need to demonstrate it with our result, we will have different metric like hit rate computation times reduction, memory overhead fitness etc

Slides 8:

Our hypothesis was that we can implement a generational caching with lru eviction strategy that can accelerate neuroevolution by 5-10% wihtouth compromising solution quality.

We predicted that the cache would demonstrates progressive learning across generations achieving a hit of 20-30% while maintaqining genetic diversity through precision controlled . Memory will remain negligible with 3-5mb

Slide 9:

Our architecture has 2 main key components,

We have the generational caching we used an Ordered dict based LRUcache from 50k to 150k enmtries, providing persistent storage across neat generation. We cahche individual neuron computations caching not just the network out put. This approach maximise cache utilization. We use fast integer hashing for O(1) operation.

Slides 11: the first shows the computations time

As we all of the cache configuration were fatser than the one without cahce

Notice the progressive learning pattern. The gap between cached and non-cached performance widens over generations, demonstrating that our cache learns and becomes more effective over time. This validates our theoretical prediction about progressive learning.

The cumulative time differences become more pronounced in later generations, showing that cache effectiveness improves as the system learns common computational patterns. This suggests even greater benefits for longer experiments

Slide 12:

Critically important - performance improvements mean nothing if we compromise solution quality.

The fitness analysis shows high variability typical of evolutionary algorithms across all configurations. Remarkably, the 50K cache achieved breakthrough performance with fitness of 1092.6, demonstrating that caching doesn't limit evolutionary potential.

Most runs converged to similar fitness ranges (180-250), showing consistent performance. Crucially, there's no systematic degradation - caching doesn't compromise solution quality.

Even more interesting, network complexity analysis reveals that caching enabled evolution of more sophisticated architectures. With 100K cache, we evolved networks with 6 nodes versus 5 nodes without cache, 444 connections versus 397 connections, and 3 layers versus 2 layers.

This represents a 20% increase in nodes, 12% more connections, and 50% more layers. Far from constraining evolution, caching actually enabled exploration of more complex network structures by reducing the computational cost penalty for evaluating sophisticated architectures

Slide 13: