HPS

Frederik Hennecke

Chess Engine Optimization in Python

Table of contents

Overview of the Project

- Developed a small chess engine from scratch.
- Designed to use the UCI protocol, enabling compatibility with platforms like Lichess, Cutechess.
- Performance-focused enhancements using state-of-the-art tools.

Motivation

- Python is popular for scientific computing, but has performance limits.
- Chess engines are computation-heavy and ideal for benchmarking.
- Goal: Optimize a Python chess engine and evaluate trade-offs.

Objectives

- Build a baseline chess engine in Python.
- Apply optimization tools:
  - JIT
  - Static compilation
  - Alternative interpreters.
- Evaluate speed (nodes/move) and playing strength (Elo).
- Analyze effort vs. performance gains.

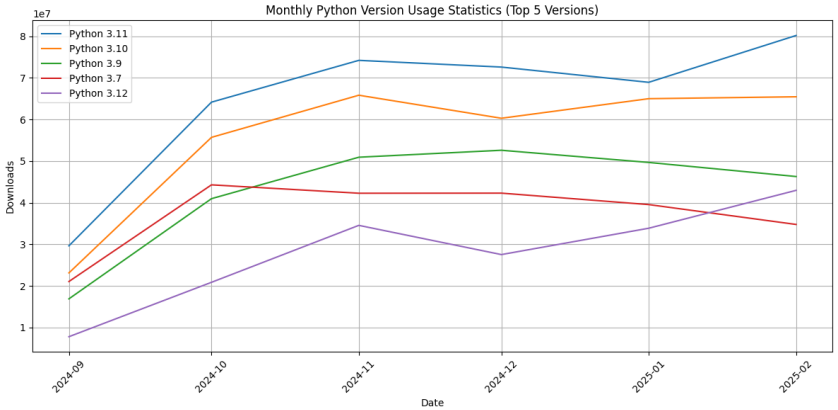
Python Bottlenecks

- Interpreter overhead
- GIL (Global Interpreter Lock)
- Dynamic typing and boxing
- Memory management

Chess Engine Components

- Main Class (UCI Interface)
- Chess Board (8x8 NumPy Array)
- Move Generator (Minimax + Alpha-Beta)
- Evaluation Function (Material + Position)

Optimization: Baseline Python 3.10



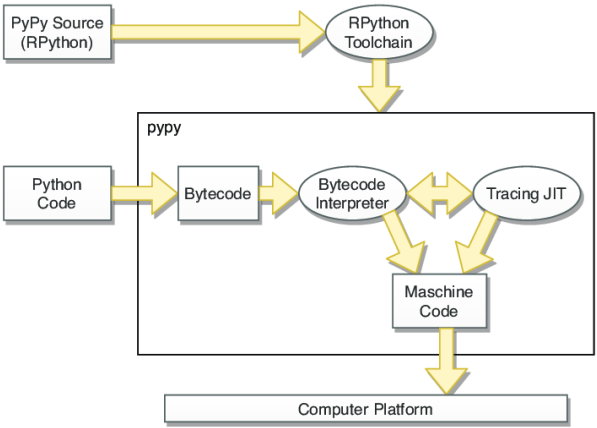
Source: *PyPI Stats*

Optimization - Python 3.12

- Drop-in upgrade using pyenv
- Bytecode and memory management improvements
- Easiest to apply
- Moderate performance boost

Optimization - PyPy

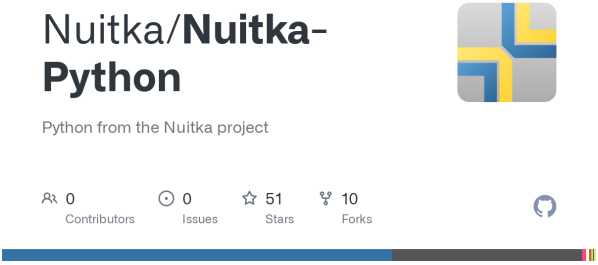
- JIT interpreter
- Boosts pure-Python code
- Slower with NumPy (cpyext bottleneck)



Source: Razik, “High-Performance Computing Methods in Large-Scale Power System Simulation”

Optimization - Nuitka

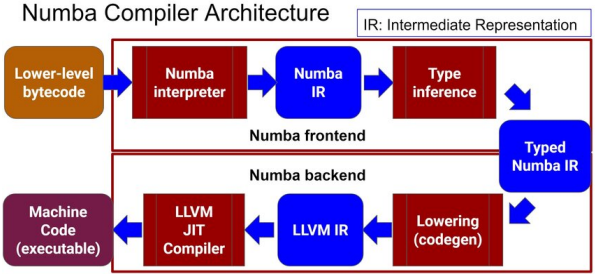
- Compiles Python to C binary
- Easy setup
- Good compatibility



Source: *This is Nuitka-Python*

Optimization - Numba

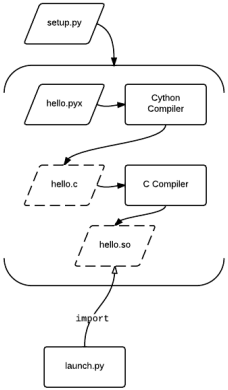
- JIT compiler with @njit
- Best for numerical/array-heavy code
- Harder to apply to object-heavy sections



Source: Saedi, “Towards Eco-Conscious Python: A Comparative Analysis of Performance, Energy Efficiency and Carbon Emissions Between CPython and Alternative Implementations”

Optimization - Cython

- Cython adds type declarations for speed
- Transpiled to C with static typing
- Multiple possible optimizations:
  - Memoryviews
  - cdef
  - cpdef
  - nogil
  - qsort



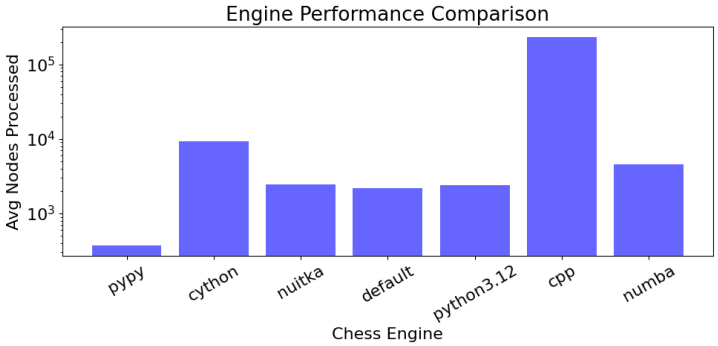
Source: DavidBrooksPokorny, *Plaques of Lambda Phages on E. coli XL1-Blue MRF*

Optimization - C++

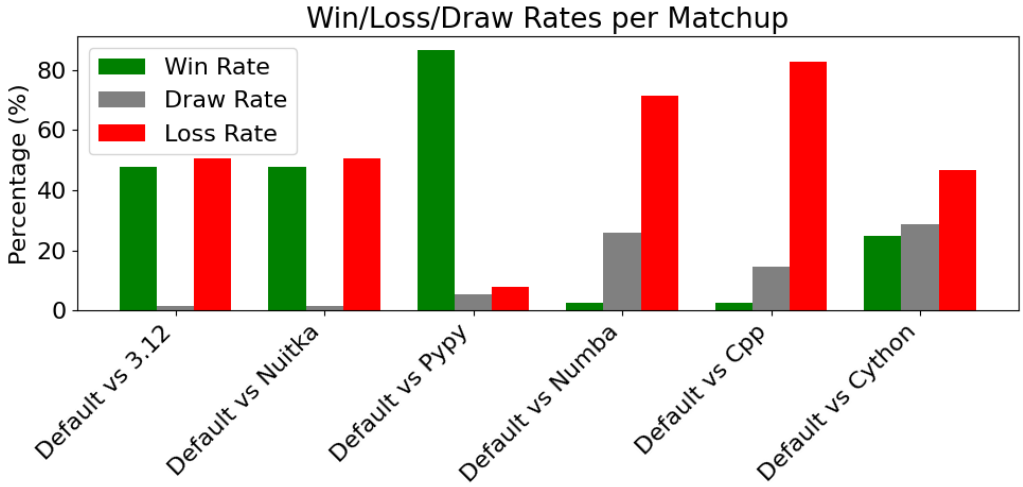
- Full engine rewritten in C++
- Cython used as a wrapper to expose C++ to Python
- Retains Python interface, gains C++ speed

Evaluation: Speed Results

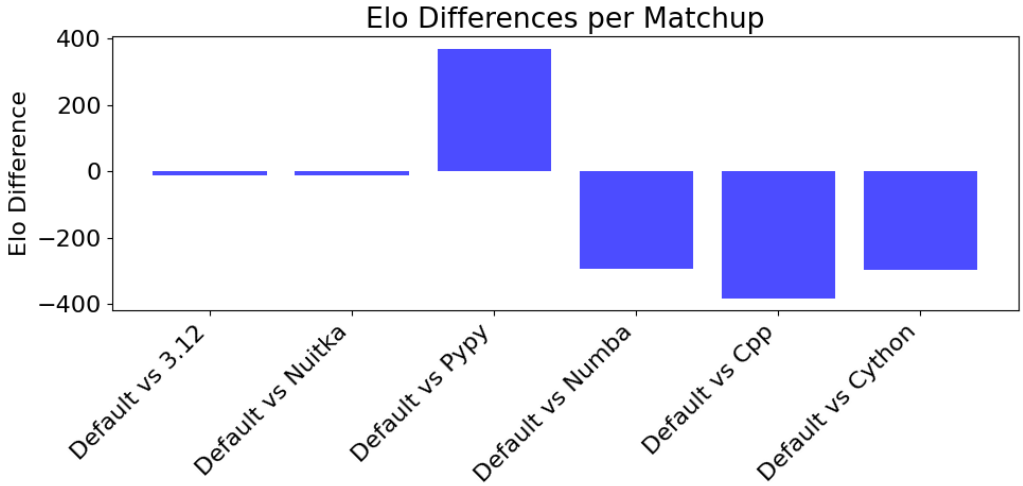
- C++: ∼231,000 nodes/sec
- Cython: ∼9,200
- Numba: ∼4,500
- Others: <∼2,500
- PyPy: ∼374 (worst)



Evaluation:Win Rate



Evaluation: Elo



Conclusion

- Key Takeaways and Insights:
  - Ease vs performance: strong correlation
  - Python 3.12, PyPy, and Nuitka are easy to use.
  - Numba is easy if only using NumPy.
  - C++ and Cython best performance but highest effort
- Future Work:
  - Explore parallelization (multiprocessing, async)
  - Other tools like Rust (via PyO3), or GPU libraries (JAX, TensorFlow)

References

DavidBrooksPokorny. *Plaques of Lambda Phages on E. coli XL1-Blue MRF*. File: LambdaPlaques.jpg. 2012. URL: https://en.wikipedia.org/wiki/Cython#/media/File:Cython_CPython_Ext_Module_Workflow.png.

PyPI Stats. URL: https://pypistats.org/.

Razik, Lukas. "High-Performance Computing Methods in Large-Scale Power System Simulation". PhD thesis. May 2020.

Saedi, Omid. "Towards Eco-Conscious Python: A Comparative Analysis of Performance, Energy Efficiency and Carbon Emissions Between CPython and Alternative Implementations". en. In: (2024). DOI: 10.13140/RG.2.2.12314.04803. URL: https://rgdoi.net/10.13140/RG.2.2.12314.04803.

This is Nuitka-Python. Python. Apr. 2025. URL: https://github.com/Nuitka/Nuitka-Python.