A Universal Library for Compressing Chess Games

CS310 @Warwick

Jaden Strudwick

The Problem

Exponential growth for online Chess platforms

- → Chess.com
- → Lichess.org

1 million games per hour [1]

Increased game data volume resulting in database scalability issues [1]

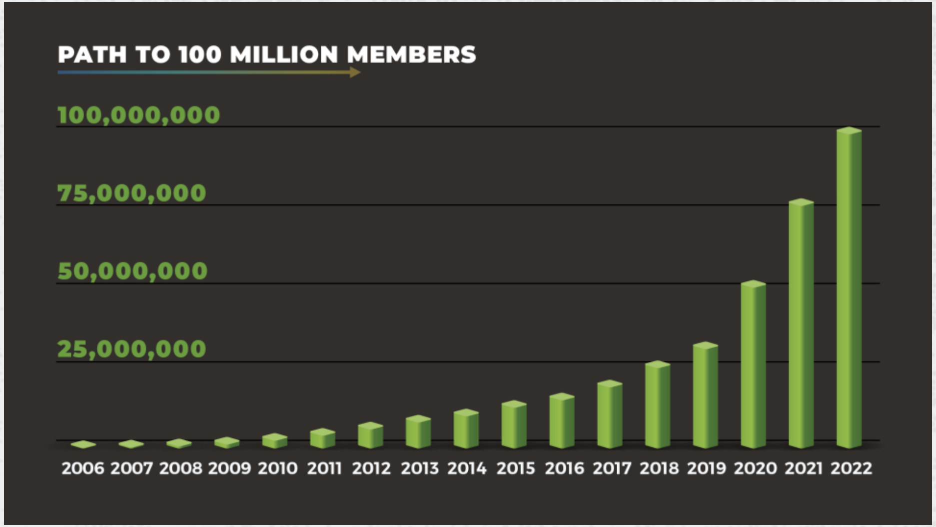
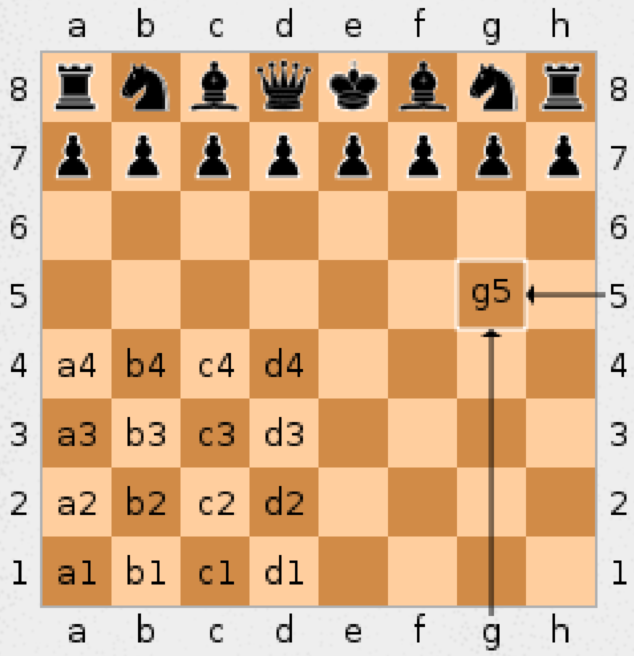


Figure 1: Growth in Chess.com Membership count [2]

[1] https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling
[2] https://www.chess.com/article/view/chesscom-reaches-100-million-members

Background: Standard Algebraic Notation (SAN)



Notation for describing chess moves

Piece Identifiers

- King: “K”
- Queen: “Q”
- Rook: “R”
- Bishop: “B”
- Knight: “N”
- Pawn: “”

Board Coordinates

- Columns (A-H) known as “Files”
- Rows (1-8) known as “Ranks”

Background: SAN Example

Identifier: N
Destination: f6

SAN: Nf6

Background: Portable Game Notation (PGN)

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 {This opening is called the Ruy Lopez.} 3... a6
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2
```

Figure 2: Example PGN file [3]

Standardized plaintext format for storing Chess Games

Developed by Steven J Edwards in 1993 [3]

Two key parts

- → Headers: Game metadata
- → Move-text: SAN moves

[3] https://en.wikipedia.org/wiki/Portable_Game_Notation

Motivation and Objectives

Storage and costs are high

- Chess.com: ~10 terabytes annually
- Reduce data volume and improve database scalability

Existing compression techniques are inaccessible

- Chess.com: Closed source
- Lichess.org: Coupled within their Java backend

Beat existing state-of-the-art solution

- Achieve lower ‘bits per move’
- Lichess has 4.46 bits per move

Universal Library for PGN Compression

- Language-agnostic
- Compress on either client/server

Existing Solutions

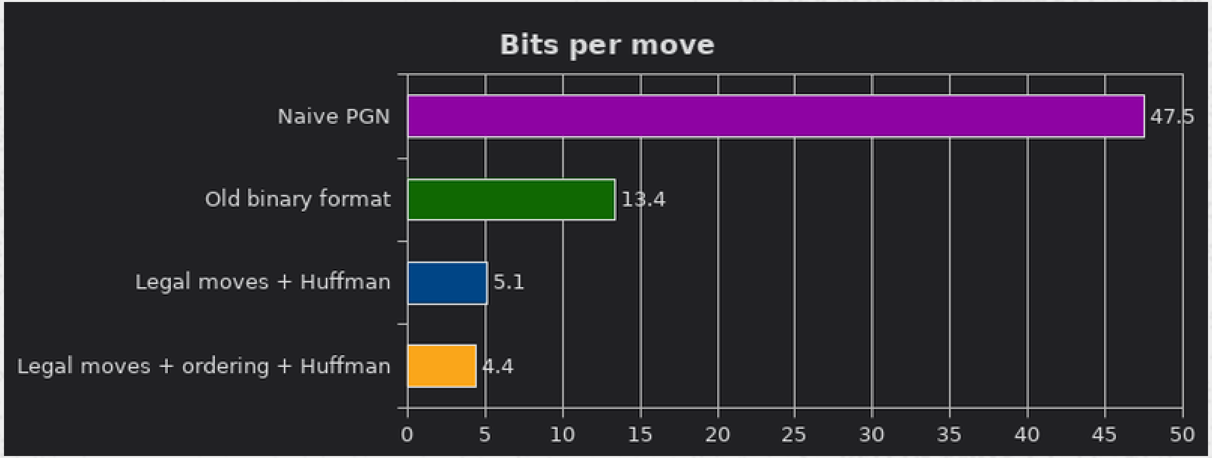


Figure 3: Comparison of Lichess' 2018 solution [4]

Only encodes moves

No PGN header compression

Not a full PGN compression algorithm

[4] https://lichess.org/@/lichess/blog/developer-update-275-improved-game-compression/Wqa7GiAA

Project Management: Tools

Rust

- → Strong type safety
- → WebAssembly compilation
- → Existing chess libraries ecosystem

WebAssembly (WASM)

- → Efficient machine-code
- → Run on any WASM runtime
- → Runtimes for C/C++, Python, .NET, etc

GitHub

- → CI/CD for running tests and code coverage
- → Version control

Project Management: Development Methodology

Test Driven Development

- → Tests before implementation
- → 96.60% code coverage

Agile

- → Sprint management each week
- → Goals are scheduled or bumped each week

Journaling

- → Markdown dev-log document
- → Updated when changes are made to the repository

Implementation: PGN Data Structure

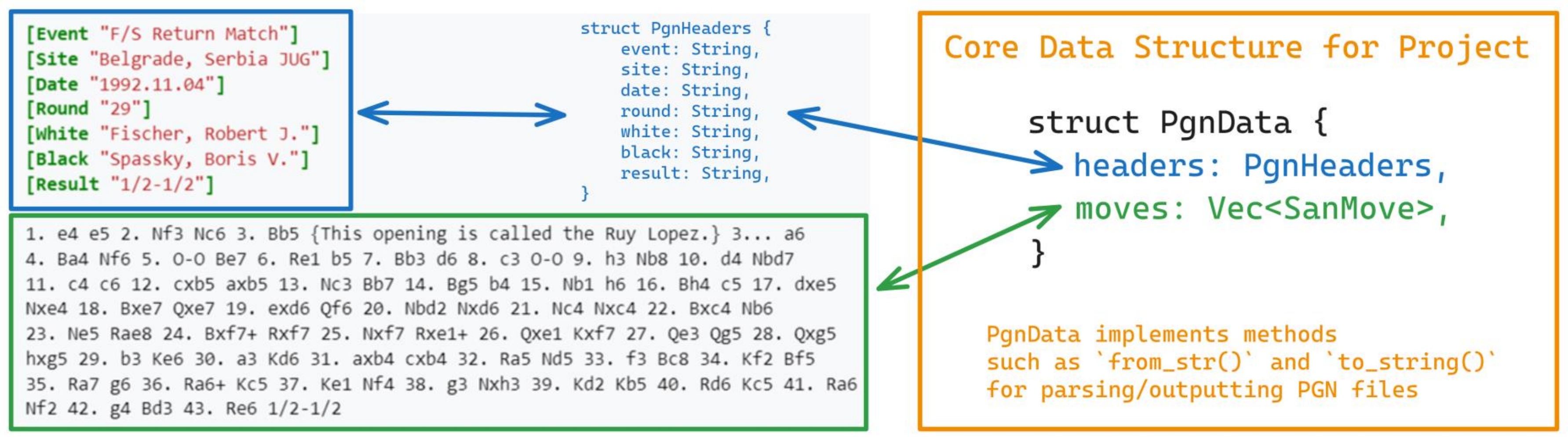


Figure 4: Composition of PgnData struct

Implementation: Overview of Architecture

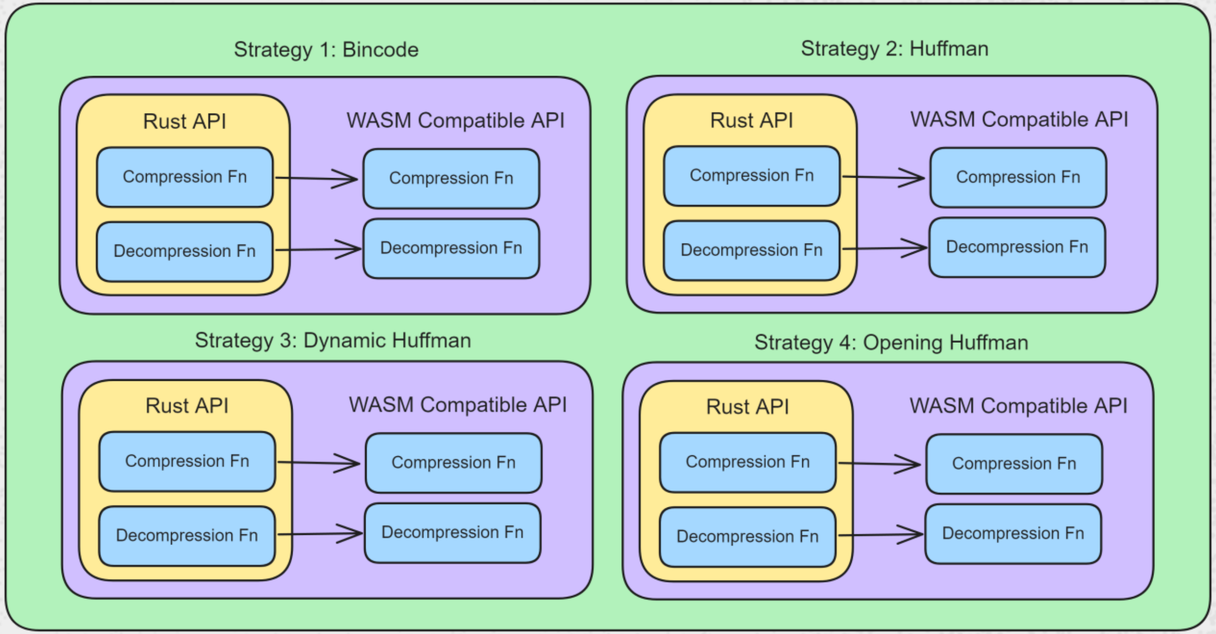


Figure 5: High-level architecture of the library

Four modules/strategies

Rust API wrapped in compatibility layer to yield WASM API

Implementation 1: Bincode

'Bincode' library for serialization of PgnData struct [5]

'Flate2' library for compressing with Zlib's DEFLATE algorithm [6]

Pros

→ Fastest compression scheme

→ Suitable for real-time data transmission

Cons

→ Serialization step not utilizing domain-specific knowledge

→ Poor compression ratios

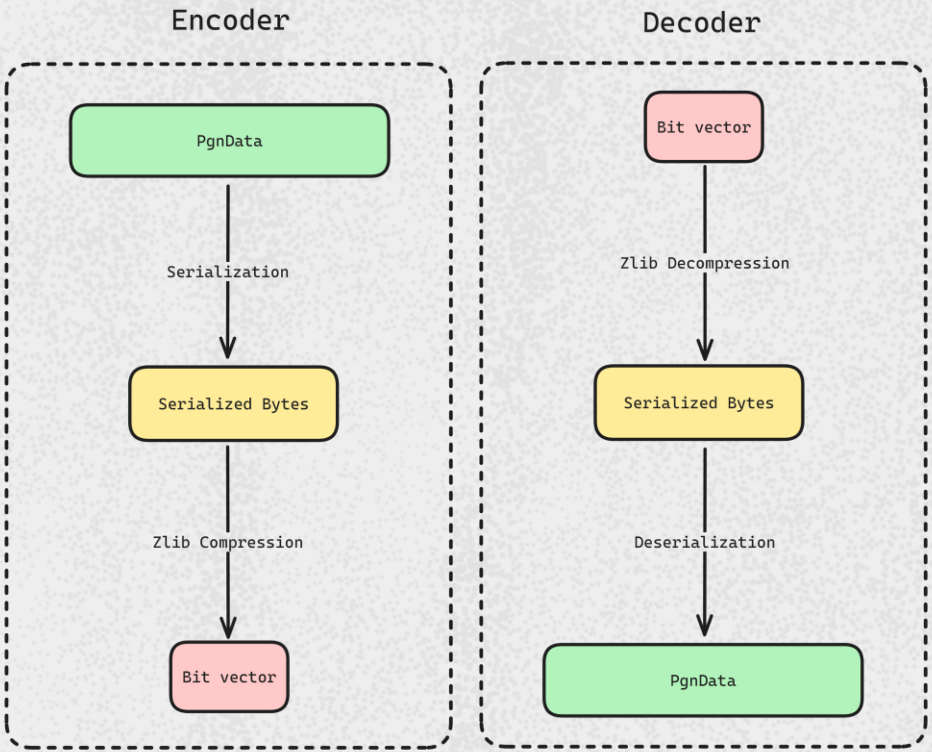


Figure 6: Business logic of Bincode encoder/decoder

[5] https://docs.rs/bincode/latest/bincode/
[6] https://docs.rs/flate2/latest/flate2/

Implementation 2: Huffman (Part 1) What is Entropy/Information Theory?

Study of quantification, storage, and communication of information [7]

Entropy measures the amount of uncertainty in a system

- → Entropy of discrete random variable X, where $p(x)$is the probability of symbol x [7]

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Based on dataset of 16 million games, entropy is 4.38 bits per move

Using entropy encoding methods, we can approach this lower entropy bound

[7] https://en.wikipedia.org/wiki/Entropy_(information_theory)

Implementation 2: Huffman (Part 2)
What is a Huffman Coding?

Entropy encoding that produces near-optimal binary codes for a set of symbols and their frequencies [8]

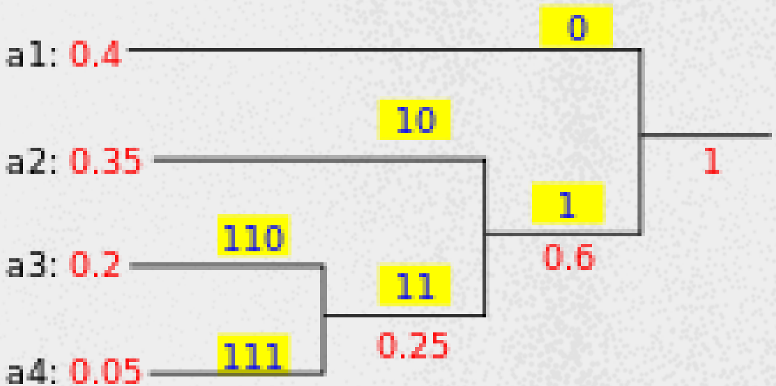


Figure 7: Huffman Coding Tree [8]

[8] https://en.wikipedia.org/wiki/Huffman_coding

Implementation 2: Huffman (Part 3)
What are our Symbols?

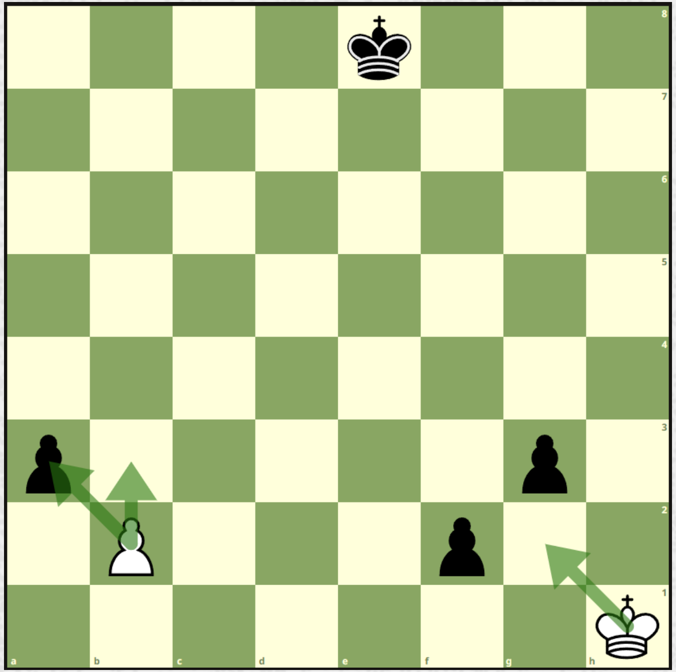
Using the SAN move (e.g. Nf6) is inefficient

Convert move to its index in the sorted list of legal moves for the current position

Strength ordered list of legal moves: [a3, Kg2, b3]

- a3 → 0
- Kg2 → 1
- b3 → 2

e4, e5, Nf3, Nc6, … ↔ 0, 3, 2, 1, …



Implementation: Huffman (Part 4)

Pros

- → Uses domain-specific knowledge
- → Approaches optimal bits per move
- → Matches Lichess's state-of-the-art solution

Cons

- → Slower than general purpose compression algorithms
- → Poor gameplay leads to large file sizes

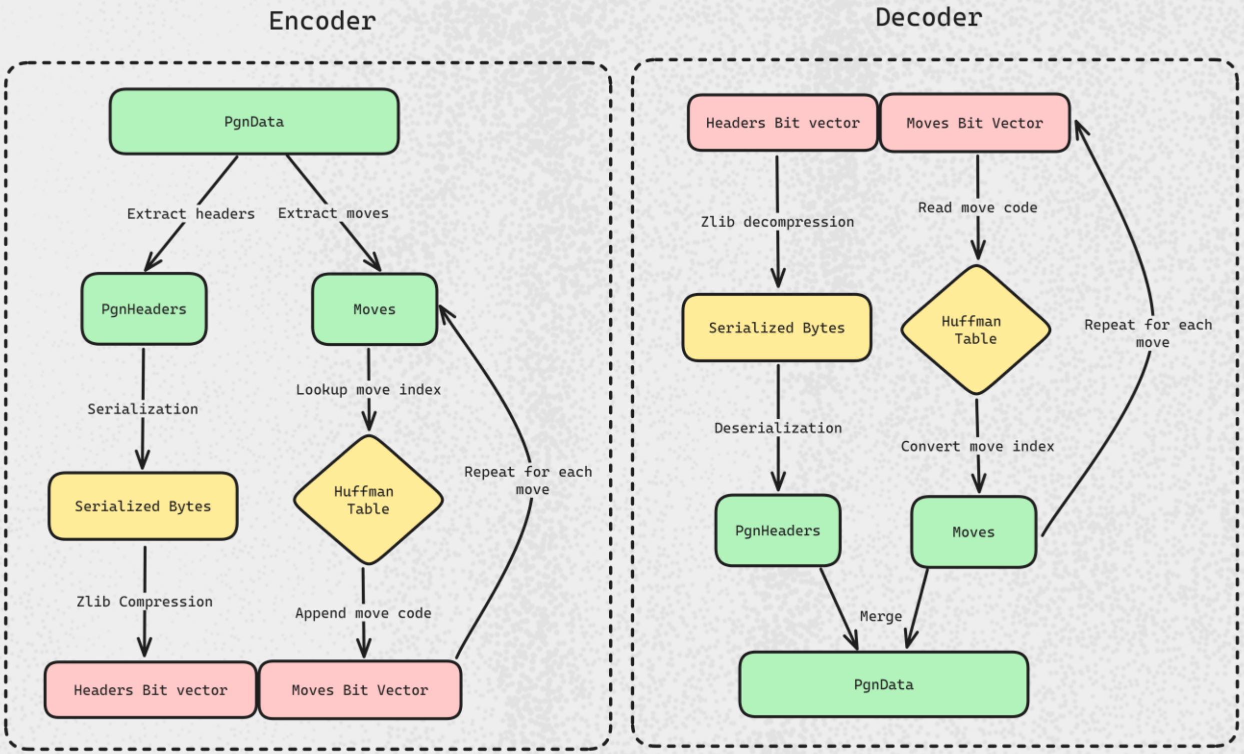


Figure 9: Business logic of Huffman encoder/decoder

Implementation: Dynamic Huffman (Part 1)

Huffman coding is a good foundation

How can we reduce uncertainty and entropy more?

Key Ideas:

→ Huffman strategy does not adapt to player

→ Any in-game trends cannot be utilized, since Huffman Coding is static

New Strategy: Update players' probability distribution after each move

Implementation: Dynamic Huffman (Part 2)

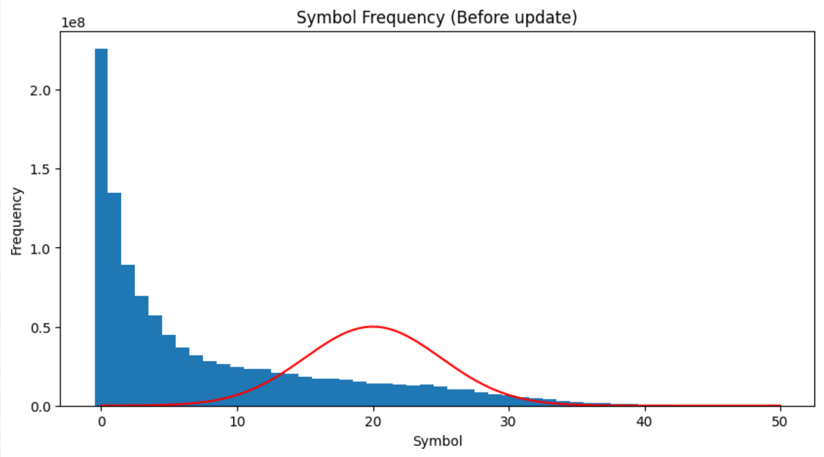


Figure 10: Symbol frequency before playing move 20

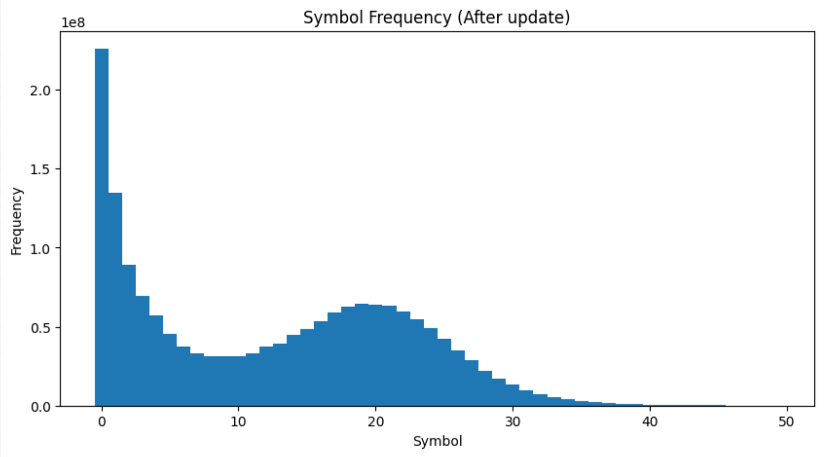


Figure 11: Symbol frequency after playing move 20

Update distributions via Gaussian after each move

$$f(x) = Height * e^{-\frac{(x-Center)^2}{2*Deviation^2}}$$

Need to find optimal Height and Deviation

Implementation: Dynamic Huffman (Part 3)

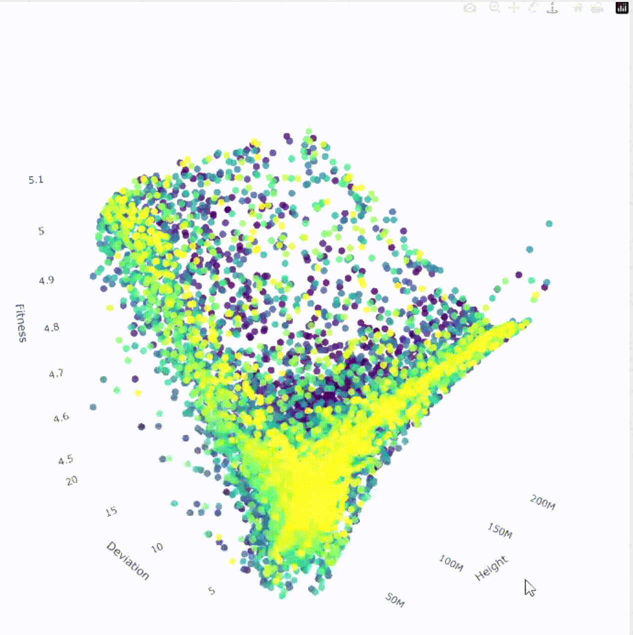


Figure 12: Population plot of GA run

Genetic Algorithm for optimal Height and Deviation

Repeatedly adjusted search space until parameters outperformed Huffman strategy

Implementation: Dynamic Huffman (Part 4)

Pros

- Surpasses Lichess's state-of-the-art solution

Cons

- Much slower than Huffman Strategy
- Less of an improvement than expected

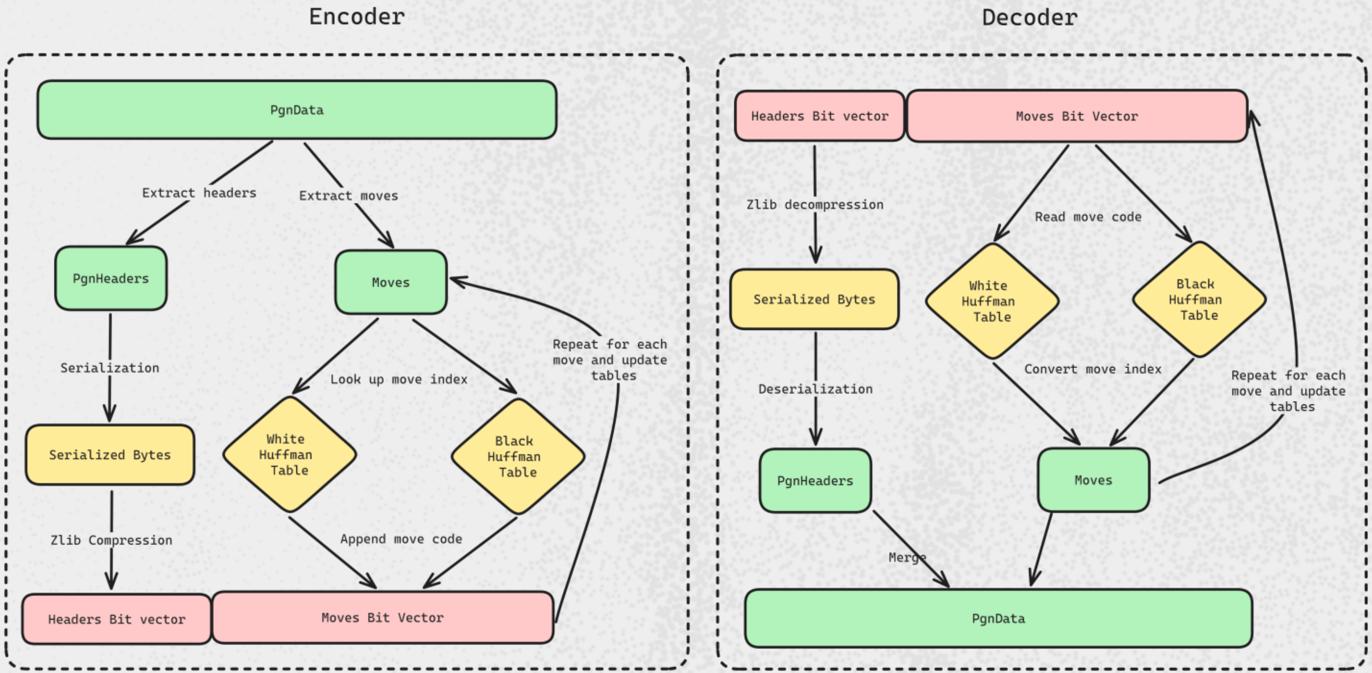


Figure 13: Business logic of Dynamic Huffman encoder/decoder

Implementation: Opening Huffman (Part 1)

Dynamic Huffman was an improvement, at the cost of speed

How else can we use chess knowledge to reduce entropy?

Key Ideas:

- → Players commonly start with a sequence of moves known as an “opening”
- → Common openings include “The Queens Gambit” and “Sicilian Defence”

New Strategy: Detect any openings and replace move sequence with the opening ID

Implementation: Opening Huffman (Part 2)

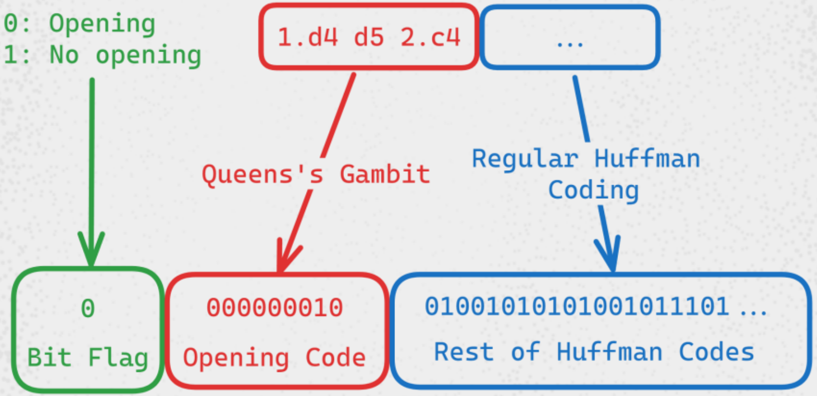


Figure 14: Sample construction of Opening Code

Use a Trie to prefix match against 512 most common openings

Take longest match, replace sequence with 9-bit ID

Bit flag to notify the decoder

Implementation: Opening Huffman (Part 3)

Pros

- → Faster than Dynamic Huffman
- → Surpasses Dynamic Huffman bits per move (and Lichess SOTA)

Cons

- → Wasted flag bit if no opening occurs in a game

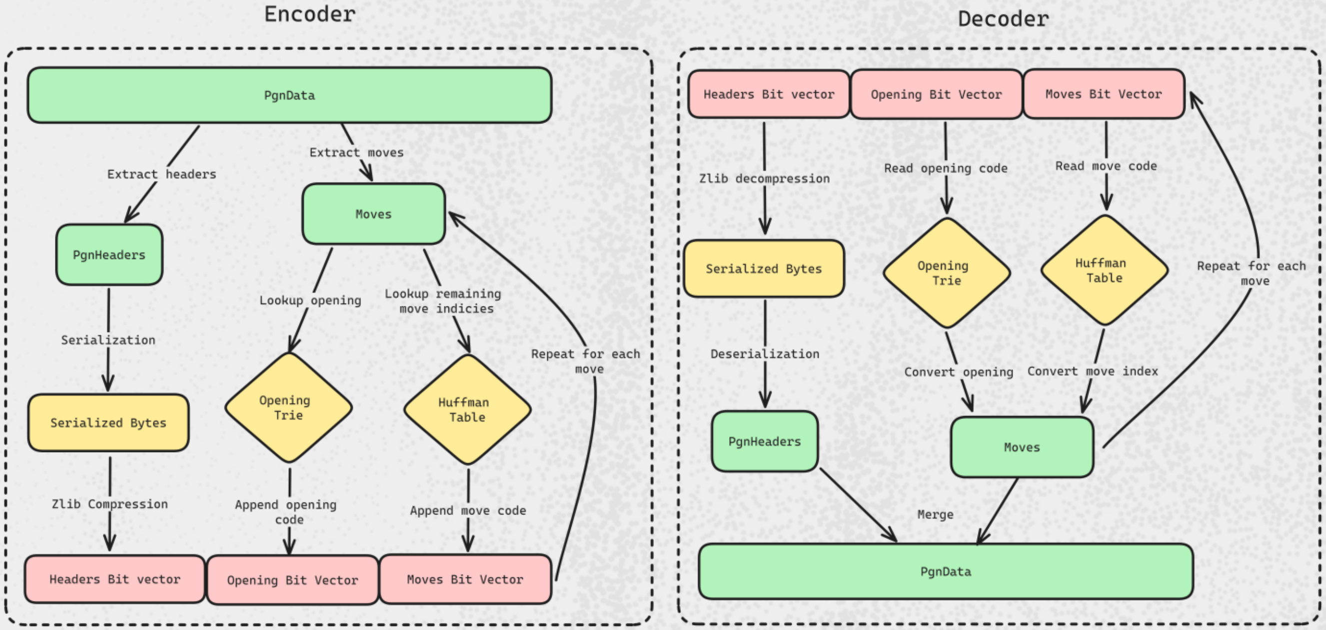


Figure 15: Business logic of Opening Huffman encoder/decoder

Implementation: Command Line Interface

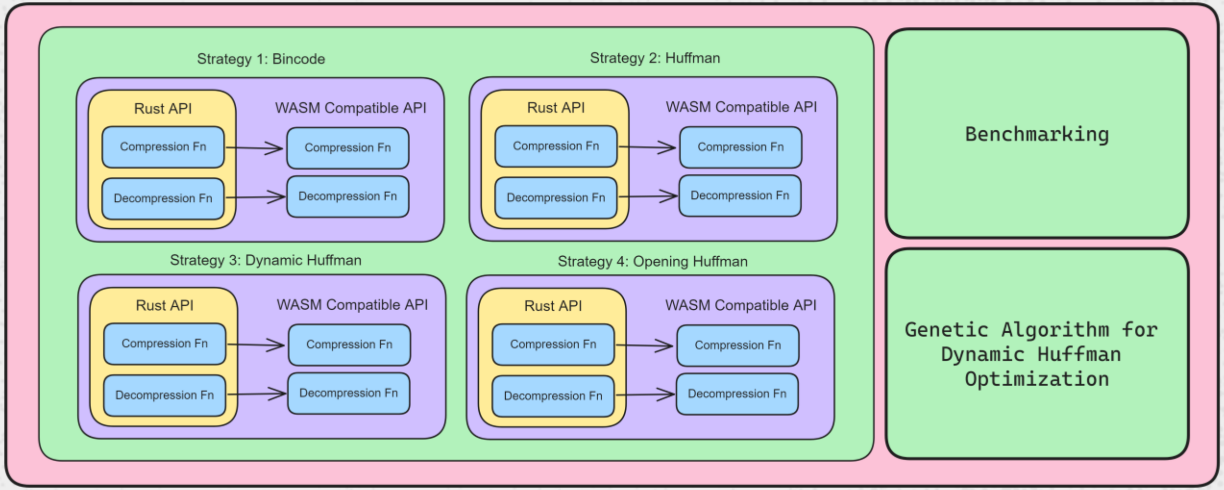


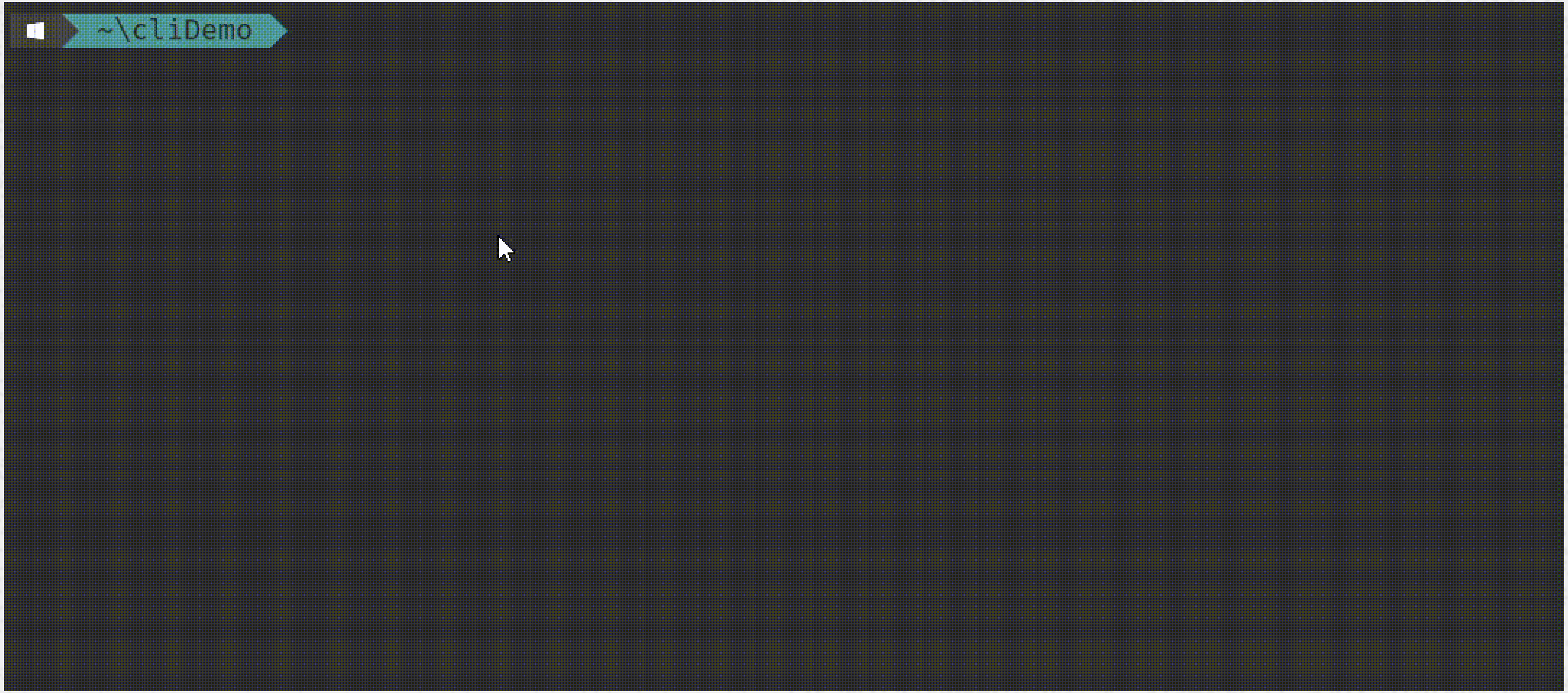
Figure 16: High-level architecture of the CLI package

CLI primarily exposes library API

Also contains

- Benchmarking tools
- Genetic Algorithm simulation for Dynamic Huffman

Implementation: Command Line Interface Demo



Implementation: Browser Extension

Proof of concept for the universal WASM library

Uses JavaScript to call the WASM API of Opening Huffman

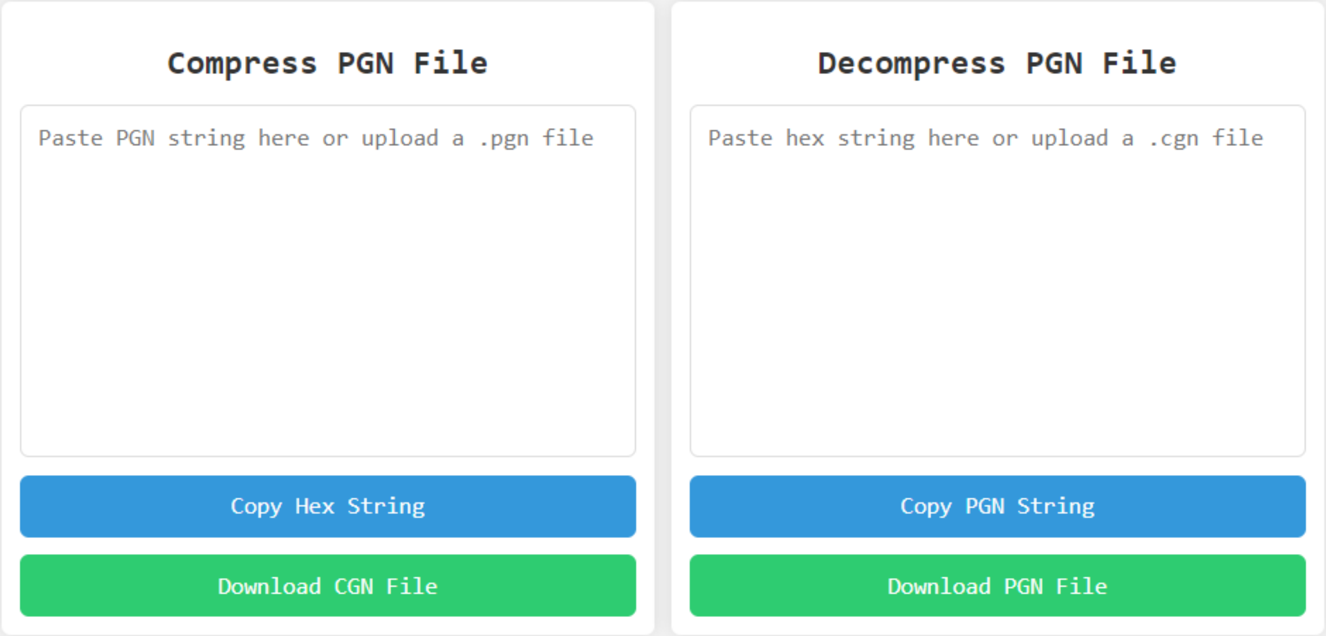
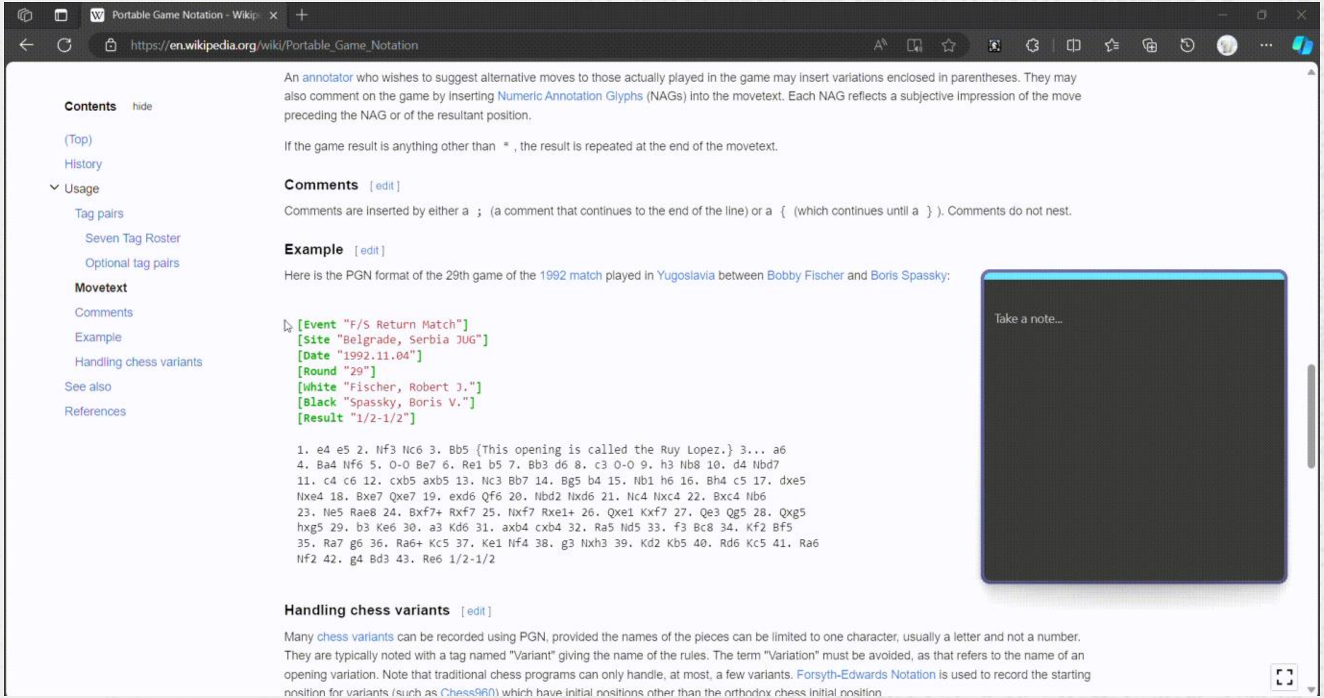


Figure 17: User-interface of the CGN Browser Extension

Implementation: Browser Extension Demo



Evaluation: Time Efficiency

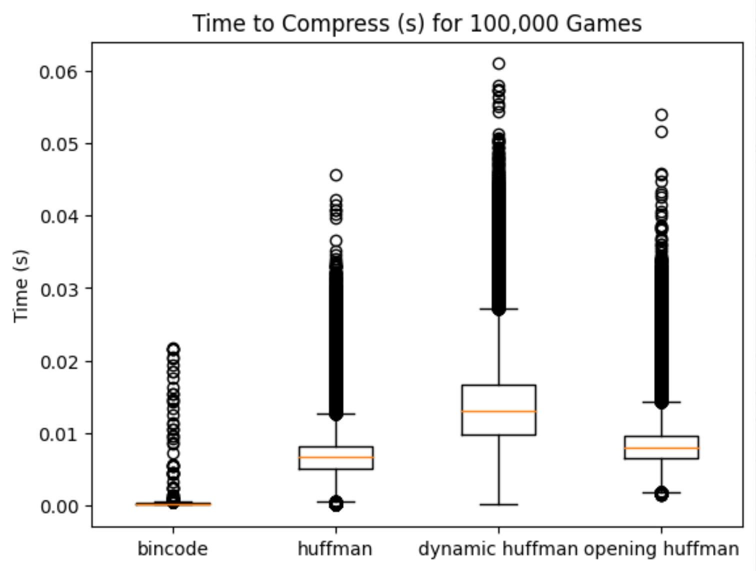


Figure 18: Compression timings for all 4 strategies

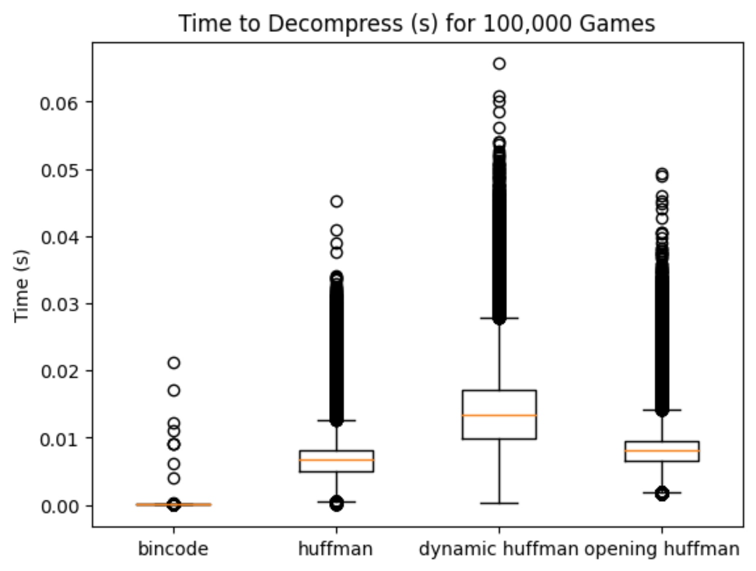


Figure 19: Decompression timings for all 4 strategies

Evaluation: Space Efficiency

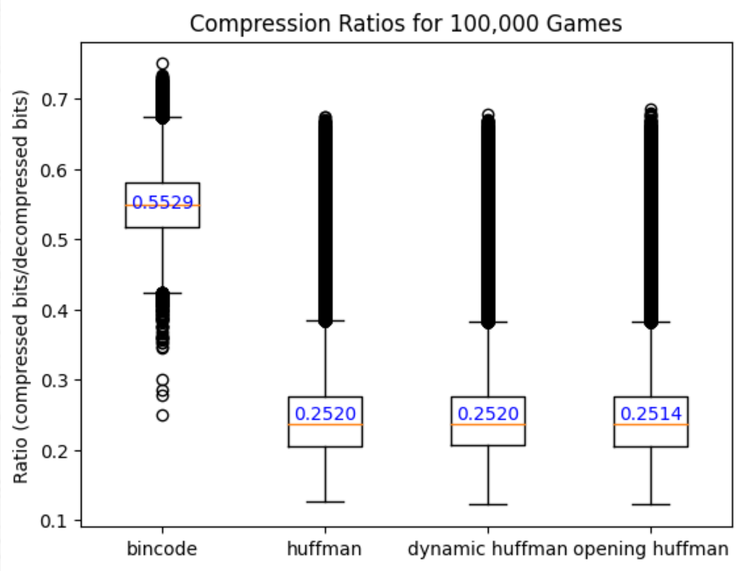


Figure 20: Compression ratios for all 4 strategies

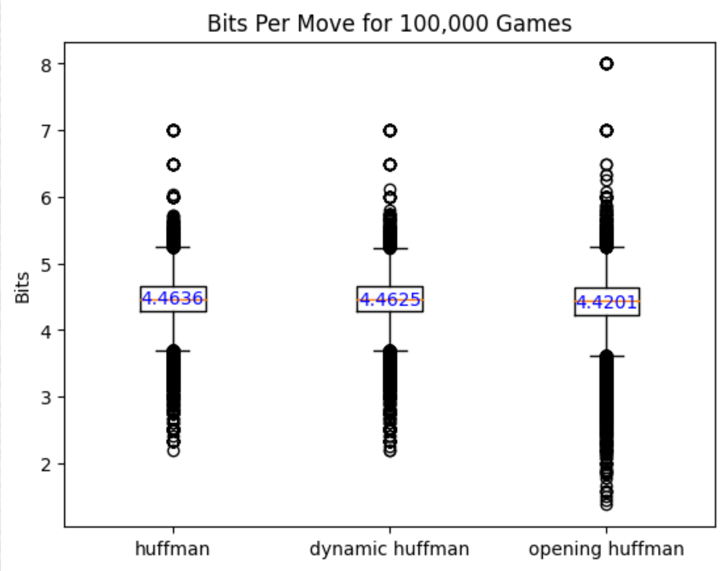


Figure 21: Bits per move for 3 Huffman strategies (Bincode excluded)

Project Outcomes and Contributions

Fulfilled goals of the Project Specification

New state-of-the-art solution via our Opening Huffman strategy

- → 1% better compression than SOTA
- → Only 0.002s slower than SOTA
- → Language agnostic

Code has been downloaded over 957 times!

- → CGN: 488 downloads
- → CGN-CLI: 320 downloads
- → CGN-JS: 149 downloads

Project Limitations

Unable to compare to Chess.com's closed source solution

Majority of compressed PGN size taken up by PGN Headers

- → **Marginal improvements to 'bits per move' are less significant**

Inefficiency of our Dynamic Huffman strategy

- → **0.02% better compression than SOTA**
- → **0.01s slower than SOTA**
- → **Overcome via Opening Huffman strategy**

Future Work

Publication of a Python package (like CGN-JS) utilizing WASM library

Investigation of other entropy-encoding methods

- **Arithmetic Coding**
- **Range Coding**

Better PGN Header management

- **Local datastore housing metadata records**
- **Compressed PGN files contain external pointers to records in datastore**

Thank you for listening

Q&A and Suggested Questions

1) How does being a ‘language agnostic’ library differ from being cross-platform?

2) Is this project more useful for chess websites like Chess.com/Lichess.org or individual Chess enthusiasts/developers?

3) You mentioned ordering potential moves by their ‘strength’ for the Huffman Coding. How is this ‘strength’ calculated?

4) How did you pick what 512 openings to include in the Opening Huffman strategy?