OopsMate; Professionalism Report

Name: Swoyam Pokharel
Student Number: 2431342
Supervisor: Prakriti Regmi
Reader: Siman Giri

Submitted On: January 20, 2026

Contents

1 · Social Aspects

1.1 · Accessibility and Democratization

The Oops!Mate chess engine is developed as an open-source project, which significantly contributes to accessibility in chess programming. Open-source software removes financial barriers and allows developers from any geographic or economic background to study, modify, and learn from the codebase.

By making the engine's source code freely available, this project helps democratize knowledge about advanced algorithmic techniques including bitboards, magic bitboards, NNUE evaluation, and alpha-beta pruning. Open-source projects promote a collaborative environment where knowledge flows freely, enabling innovation and learning opportunities that would otherwise be restricted.

1.2 · Educational Impact

The primary educational contribution of this project lies in its comprehensive documentation and transparent development process. Unlike production chess engines such as Stockfish, which have evolved over decades with contributions from thousands of developers, Oops!Mate is built from scratch with each component carefully documented and explained.

This approach directly addresses a gap in chess programming education. While engines like Stockfish and Leela Chess Zero are open source, their massive codebases and complex architectures can be overwhelming for newcomers.

The educational value has been validated through practical engagement. As another developer recently contacted via LinkedIn, expressing appreciation for the project blog and interest in learning chess programming through the documented process. This demonstrates that the project serves as an accessible educational resource for atleast some other developer out there.

Furthermore, by benchmarking each incremental improvement, such as the addition of alpha-beta pruning, move ordering heuristics, or NNUE evaluation, the project provides empirical data about the relative importance of different techniques. This quantitative approach to understanding chess engine components is rarely documented in accessible formats.

1.3 · Community Contribution

Furthermore, this engine has a symbolic significance, this is one of, if not the first engine thats publicly available, and open sourced from Nepal. However this aspect should be viewed as a contribution to the local tech market rather than a transformative achievement. The global chess programming community is well established, with active forums such as TalkChess and open-source projects like the Chess Programming Wiki providing extensive knowledge resources (Chess Programming Wiki, 2024).

The project's contribution lies more in adding one more accessible learning resource to this existing ecosystem rather than pioneering entirely new territory.

2 · Ethical Aspects

2.1 · Fair Play and Potential Misuse

A significant ethical concern with any chess engine is its potential misuse for cheating in online chess platforms. Chess.com and Lichess have reported increasing incidents of computer assisted cheating, where players use engines during games to gain unfair advantages.

Oops!Mate, once it reaches its target strength of 1800 Elo or higher, could be misused in this manner. However, several factors mitigate this concern:

1. **Detection mechanisms**: Major chess platforms employ sophisticated statistical analysis and behavioral detection systems to identify engine assistance.
2. **Relative strength**: With a target Elo of 1800, Oops!Mate would be significantly weaker than existing engines like Stockfish (3600+ Elo), making it a less attractive tool for cheating compared to readily available alternatives.
3. **Educational purpose**: The project's transparent documentation and educational focus may actually help chess platforms and anti-cheat researchers understand engine behavior better.

Once the engine is fully complete, it will also include a disclaimer against unethical use, but with that said, the ultimate responsibility lies with users.

2.2 · Transparency and Open Source Development

The decision to develop Oops!Mate as an open-source project aligns with ethical principles of transparency in AI development. Unlike proprietary engines where evaluation methods and search algorithms remain opaque, open-source engines allow complete insight of decision-making processes

This transparency serves multiple ethical purposes:

1. **Reproducibility**: Other individuals can verify claims about the engine's strength and the effectiveness of different techniques.
2. **Educational integrity**: Students and learners can understand exactly how the engine works rather than treating it as a "black box"
3. **Collective improvement**: The chess programming community can identify bugs, suggest improvements, and learn from the implementation.

2.3 · Training Data Ethics

The engine plans to implement Efficiently Updatable Neural Networks (NNUE) for position evaluation. Rather than training a neural network from scratch, which would require generating billions of positions through self-play or collecting human game data with potential copyright and privacy concerns; this project uses pre-trained NNUE networks from Stockfish.

Stockfish's NNUE networks are trained on positions generated through the engine's own self-play, avoiding ethical concerns related to:

1. **Data ownership**: No human game data is used without consent.
2. **Privacy**: No personal information from human players is incorporated.
3. **Copyright**: Training data is generated by the engine itself rather than scraped from proprietary databases.

Using Stockfish's pre-trained networks is ethically sound as Stockfish is licensed under GPL v3, explicitly permitting such reuse (Stockfish Team, 2024). This approach follows the principle of building upon existing open-source work, as advocated by the Free Software Foundation.

3 · Legal Aspects

3.1 · Open Source Licensing

The project will be released under an open-source license, though the specific license has not yet been finalized. The options that are being considered are:

1. **GPL v3**: This copyleft license would require derivative works to also be open source, following Stockfish's example.
2. **MIT License**: A permissive license allowing commercial use with minimal restrictions.
3. **Apache 2.0**: Provides patent protection while remaining permissive.

The choice of license has significant legal implications for how others can use and distribute the engine. GPL v3 ensures the engine and its derivatives remain open source, while MIT or Apache licenses allow commercial use without requiring source code disclosure (Free Software Foundation, 2007; Open Source Initiative, 2024).

3.2 · Intellectual Property

Chess algorithms are generally not subject to patent protection, as most fundamental techniques (minimax, alpha-beta pruning, bitboards) have been in the public domain for decades. More recent innovations like NNUE have been published in academic contexts and implemented in open-source projects without patent restrictions.

The project does not incorporate any proprietary algorithms or patented techniques. All implemented methods are based on well-established chess programming literature and open source implementations.

3.3 · Third-Party Dependencies

The Rust implementation of Oops!Mate has minimal external dependencies. The primary dependency is the `rand` crate, used for random number generation in certain engine features. The `rand` crate is licensed under MIT/Apache 2.0, which permits use in both open-source and commercial projects without legal restrictions.

Rust itself, as the programming language, is dual-licensed under MIT and Apache 2.0 licenses, ensuring no legal barriers to distribution.

3.4 · UCI Protocol Implementation

The Universal Chess Interface (UCI) is an open communication protocol developed by Rudolf Huber and Stefan Meyer-Kahlen for chess engines. The protocol specification is freely available and has no copyright restrictions on implementation.

There are no known legal implications of implementing UCI in an open-source engine. Major chess engines including Stockfish, Leela Chess Zero, and Komodo all implement UCI without legal issues. The protocol's openness has made it the de facto standard for chess engine communication.

3.5 · Code Attribution and Originality

All code in Oops!Mate is written from scratch. The project properly cites all algorithmic techniques in its literature review and documentation, attributing ideas to their original researchers (e.g., Shannon for minimax, Knuth and Moore for alpha-beta, Nasu for NNUE). This follows academic standards for intellectual attribution while maintaining code originality.

3.6 · Trademark Considerations

A preliminary search for "OopsMate" or "Oops!Mate" revealed minimal existing use:

1. One YouTube channel with 73 subscribers uses "OopsMate" as its name.
2. No registered trademarks were found in the UK Intellectual Property Office database nor any other sources.
3. No chess-related software or services use this name.

Given the limited existing use and the different context (YouTube channel vs. chess engine), trademark conflicts appear unlikely. However, the project does not claim exclusive rights to the name and operates purely as an educational open-source project rather than a commercial product.

4 · Security Aspects

4.1 · Input Validation and FEN Parsing

The engine accepts Forsyth-Edwards Notation (FEN) strings to set up board positions. While FEN parsing inherently involves string processing, the security implications are minimal for several reasons:

1. **Type safety**: Rust's strong type system prevents common vulnerabilities like buffer overflows that plague C/C++ implementations.
2. **Validation**: The FEN parser includes validation logic to reject malformed strings, preventing undefined behavior.
3. **Local execution**: The engine runs locally without network exposure, limiting attack surface.

Malformed FEN strings can cause parsing errors but cannot exploit memory vulnerabilities due to Rust's safety guarantees.

4.2 · Memory Safety

One of Rust's primary advantages over traditional chess engine languages (C and C++) is its memory safety guarantees. The Rust compiler enforces ownership and borrowing rules at compile time, preventing entire classes of vulnerabilities :

1. **No buffer overflows**: Array bounds are always checked.
2. **No use-after-free**: The ownership system prevents accessing deallocated memory.
3. **No null pointer dereferences**: Rust uses the `Option` type instead of nullable pointers.
4. **No data races**: The type system prevents concurrent access violations.

Although Rust does not provide automatic garbage collection. Instead, it uses deterministic memory management through its ownership system, which is more efficient for real-time systems like chess engines where predictable performance is critical. This makes any memory based vulnerabilities highly unlikely.

4.3 · Denial of Service Prevention

Unlike networked chess engines, Oops!Mate operates entirely locally without internet connectivity or network interfaces. This eliminates common DoS attack vectors:

1. **Network floods**: Not applicable as the engine does not open network sockets.
2. **Resource exhaustion**: The engine can only consume resources on the local machine where it's executed.

Potential resource exhaustion scenarios include:

1. **Infinite loops**: Mitigated by careful algorithm implementation and testing.
2. **Excessive memory use**: The transposition table has fixed size limits.
3. **Time management**: The UCI protocol includes time control commands that the engine respects.

4.4 · Supply Chain Security

The project has a minimal dependency footprint, reducing supply chain risk:

1. **Rust toolchain**: Distributed by the Rust Foundation through official channels with cryptographic verification.
2. **rand crate**: A widely-used, well-maintained library with thousands of dependents, and is unlikely to be vulnerable

4.5 · Open Source Security Benefits

Since the engine is completely opensource, this enables collective security review by the broader community. This transparent approach allows independent verification of the implementation's security properties. Security conscious users can conduct their own code audits to assess potential vulnerabilities before integrating the engine into their systems, supporting informed risk assessment.