Blunder Guard: Efficient Small Language Models for Heuristic-Guided Chess Commentary Generation

Seyed Ali Hosseininasab
Shahid Beheshti University
Tehran, Iran
seyed123ali123@gmail.com

Farima Kafi*
Shahid Beheshti University
Tehran, Iran

Abstract

We present **Blunder Guard**, a locally deployable system for generating pedagogical chess commentary by integrating a domain-adapted Small Language Model (SLM) with classical engine evaluation. Unlike prior approaches relying on large-scale LLMs or supervised datasets of annotated games, Blunder Guard demonstrates that a 1.7B-parameter Qwen3 model—fine-tuned on unstructured classical chess literature—can generate strategic, human-understandable commentary when guided by heuristics derived from Stockfish.

The training corpus comprises over 2.8M tokens extracted from canonical chess books using a parallelized PDF extraction pipeline. Positional features are computed from Stockfish evaluations and processed through a structured feature extractor before being fed into the SLM, enabling commentary that aligns with objective position assessments. The model is quantized to GGUF (Q4_K_M) and deployed with `llama.cpp`.

Evaluation by chess experts indicates that the system produces commentary that is intelligible, strategically accurate, and pedagogically useful, even on previously unseen positions. Figure 1 illustrates the interactive interface, highlighting real-time board analysis and commentary generation. All code and models are publicly available.

Keywords

Chess AI, Small Language Models, Explainable AI, Natural Language Generation

1 Introduction

The confluence of superhuman chess engines and natural language generation presents a unique opportunity for chess education: explaining *why* a position favors one side in terms accessible to human learners. Modern engines such as Stockfish [10] and AlphaZero [9] achieve tactical precision far beyond human masters, yet their output—centipawn evaluations and principal variations—remains opaque to intermediate players. Conversely, Large Language Models (LLMs) generate fluent explanations but frequently hallucinate tactical motifs and misevaluate positions, rendering them unreliable for serious instruction.

This work addresses the problem of **interpretable chess analysis**: generating commentary that is simultaneously (1) strategically accurate, grounded in objective engine evaluation; (2) pedagogically structured, following the conventions of classical chess literature; and (3) computationally efficient, enabling fully local deployment without cloud dependencies.

*Instructor: Maryam Tahmasbi

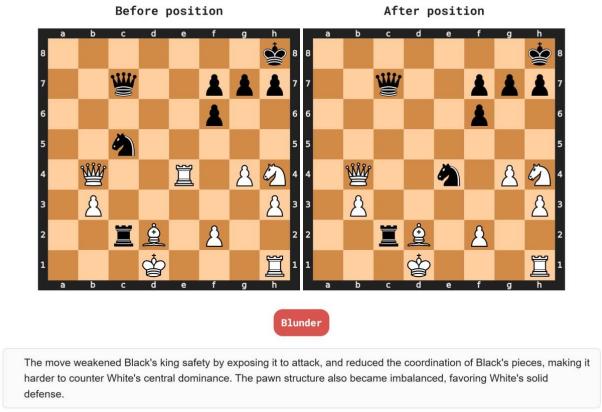

Figure 1: Blunder Guard user interface showing interactive chessboard, real-time position analysis, and generated commentary for moves before and after evaluation.

Existing approaches fall into two categories. Supervised methods [5] require large annotated datasets and lack scalability. LLM-based methods [6] achieve quality through scale but incur prohibitive costs and raise privacy concerns. Neither approach satisfies the needs of clubs, schools, or individual players seeking affordable, private coaching tools.

We propose a different paradigm: fine-tuning Small Language Models (SLMs) on unstructured chess literature while guiding generation with engine-verified heuristics. Canonical instructional texts encode reusable explanatory patterns without position-specific labels. By learning these patterns, an SLM can generalize to novel positions when guided by heuristic priorities derived from Stockfish, producing commentary that is both informative and pedagogically sound.

2 Related work

The intersection of artificial intelligence and chess has long served as a testbed for exploring interpretable decision-making and natural language generation. While early approaches focused primarily on playing strength [2, 9], recent research has shifted toward making strong chess engines comprehensible to human players. This evolution reflects a broader trend in explainable AI: moving from black-box superhuman performance to systems that can articulate *why* certain decisions are made. In this context, chess commentary generation has emerged as a challenging benchmark, requiring models to balance strategic accuracy with pedagogical clarity. Unlike generic text generation, chess commentary demands deep

domain knowledge, precise move annotation, and the ability to convey complex positional concepts in accessible language. In this section, we review two closely related works that address these challenges through distinct methodological lenses: concept-guided explanation frameworks and neural approaches to grounded commentary generation. These works inform our approach of combining lightweight language models with traditional engine analysis for efficient, locally-deployable chess coaching systems.

2.1 Bridging the Gap between Expert and Language Models

Kim et al. [6] address a fundamental challenge in explainable artificial intelligence: bridging the gap between expert decision-making models and large language models (LLMs). While expert models such as modern chess engines achieve superhuman performance in complex decision-making tasks, their outputs are difficult for humans to interpret. In contrast, LLMs are capable of generating fluent and natural language explanations but often suffer from hallucinations and lack domain-specific reasoning capabilities. The authors study this problem in the context of chess commentary generation, which serves as a representative task for explaining complex decisions through natural language.

To overcome these limitations, the paper proposes a framework called *Concept-guided Chess Commentary* (CCC). The key idea is to extract high-level chess concepts—such as king safety, passed pawns, mobility, and threats—from a neural chess expert model using concept-based explanation techniques. These concepts are represented as vectors learned via linear classifiers trained on positions labeled by a classical chess engine. For a given move, the framework prioritizes concepts by comparing their relevance before and after the move, identifying which concepts are most influential in explaining the decision.

The prioritized concepts are then provided as guidance to a large language model during commentary generation. This integration allows the LLM to focus on strategically important aspects of the position while avoiding incorrect or hallucinated explanations. As a result, the generated commentary is both linguistically fluent and strategically accurate.

In addition to commentary generation, the authors address the problem of evaluation. They introduce *GPT-based Chess Commentary Evaluation* (GCC-Eval), a multi-dimensional evaluation framework that assesses commentary based on relevance, completeness, clarity, and fluency. Unlike traditional similarity-based metrics such as BLEU or ROUGE, GCC-Eval incorporates expert chess knowledge and shows a significantly higher correlation with human expert judgments.

Experimental results on a standard chess commentary dataset demonstrate that the proposed CCC framework outperforms existing baselines and even rivals human-generated commentary in terms of informativeness and linguistic quality. Overall, this work highlights the effectiveness of concept-based explanations as an interface between expert models and language models, and it provides a generalizable approach for interpretable decision explanation beyond the domain of chess.

2.2 Learning to Generate Move-by-Move Chess Commentary

Jhamtani et al. [5] study the task of automatically generating natural language commentary for individual moves in chess games. The authors frame this problem as a grounded natural language generation task, where the generated text must accurately reflect the game state, adhere to the rules of chess, and capture the strategic and pragmatic motivations behind a move. Chess commentary generation is particularly challenging due to the need for domain knowledge, the evolving nature of the game state, and the high variability in how humans describe the same move.

A major contribution of this work is the introduction of a large-scale chess commentary dataset collected from online chess forums. The dataset contains more than 298,000 move–commentary pairs spanning over 11,000 games, making it the first dataset of this scale for move-level chess commentary generation. An analysis of the data reveals substantial diversity in commentary styles, which the authors categorize into several types, including direct move descriptions, evaluations of move quality, comparisons with alternative moves, strategic planning explanations, contextual game-level information, and general comments. This diversity highlights the importance of content selection and pragmatic reasoning in the generation process.

To address these challenges, the authors propose a neural model called *Game-Aware Commentary* (GAC). The model is trained end-to-end and incorporates multiple sources of domain-specific information extracted from the chess board state before and after a move. These features include move-related information (e.g., piece type and position), threat-related information (e.g., attacking and defending pieces), and score-based features obtained from a chess engine to estimate move quality. Feature representations are embedded in a shared continuous space, and a bidirectional LSTM encoder is used to model feature conjunctions. A selection mechanism is further employed to dynamically identify salient features during text generation, followed by an LSTM decoder that produces the commentary.

Experimental results show that the proposed GAC model outperforms several baselines, including template-based methods, nearest-neighbor approaches, and models that rely solely on raw board representations. Although automatic metrics such as BLEU yield relatively low scores due to the inherent variability of natural language commentary, human evaluation demonstrates that the generated commentaries are comparable to, and in some cases indistinguishable from, human-written commentary in terms of correctness and fluency. Overall, this work establishes a strong foundation for research on grounded and interpretable language generation in games and serves as a key reference for subsequent studies in chess commentary and explainable decision-making systems.

3 Dataset Preparation

In this section, we describe the construction of our training corpus and evaluation framework. Our approach diverges from conventional supervised learning: we employ a two-stage pipeline where (1) a Small Language Model (SLM) is first fine-tuned on pure

chess literature without position-specific labels, and (2) a heuristic-enhanced evaluation system is subsequently applied to guide inference. This design choice prioritizes general strategic understanding over memorization of specific position-label pairs.

3.1 Training Corpus Composition

The training dataset comprises curated classical chess literature representing diverse pedagogical approaches and historical periods. Unlike prior work that relies on annotated game databases with move-level labels [5], our corpus consists of unstructured instructional text authored by renowned chess masters.

3.1.1 Chess Literature Corpus. We digitized five canonical chess instruction books:

- *The Game of Chess* by Siegbert Tarrasch (1935) [12]: Foundational strategic principles and systematic piece development
- *Zurich International Chess Tournament, 1953* by David Bronstein (1979) [1]: Deep annotations of candidate tournament games
- *The Life and Games of Mikhail Tal* by Mikhail Tal (1997) [11]: Attacking patterns and dynamic piece play
- *How to Reassess Your Chess* by Jeremy Silman (2010) [8]: Positional imbalances and modern strategic thinking
- *Logical Chess: Move by Move* by Irving Chernev (1957) [3]: Pedagogical explanations for intermediate players

These texts span classical principles [12], modern imbalance theory [8], tactical brilliance [11], and instructional methodology [3], providing comprehensive coverage of chess pedagogy.

3.1.2 Text Extraction and Preprocessing. Source materials existed in PDF format, necessitating robust extraction and cleaning. We developed a parallelized pipeline implemented in Python using `pypdf` for PDF parsing. Algorithm 1 describes the extraction process.

Algorithm 1 PDF Text Extraction Pipeline

Require: PDF directory D_{pdf}, Output directory D_{txt}
Ensure: Clean text files T
$F \leftarrow \text{ListPDFs}(D_{pdf})$ ▷ Enumerate PDF files
$T \leftarrow \emptyset$
for each file $f \in F$ **in parallel** using `ThreadPoolExecutor` **do**
  $text \leftarrow \text{ExtractText}(f)$ ▷ `PdfReader` page extraction
  $text \leftarrow \text{RemoveHyphenation}(text)$ ▷ Join broken words
  $text \leftarrow \text{NormalizeWhitespace}(text)$ ▷ Collapse multiple spaces/newlines
  if $\text{Length}(text) > \theta_{min}$ **then** ▷ Filter empty/low-quality extractions
    $t_{path} \leftarrow \text{Save}(text, D_{txt}, \text{ChangeExtension}(f, .txt))$
    $T \leftarrow T \cup \{t_{path}\}$
  end if
end for
return T

The implementation handles critical PDF artifacts: line-break hyphenation (e.g., "posi- tion" → "position") and irregular whitespace from column layouts. Parallel processing utilizes all available CPU cores for efficient batch processing of the corpus. Total extracted text: approximately 2.8 million tokens after cleaning and deduplication.

3.2 Heuristic Enhancement Module

Following model training, we employ a heuristic module to guide commentary generation during inference. This module bridges the gap between the model's general strategic knowledge and specific position evaluation, analogous to the concept extraction approach of Kim et al. [6] but implemented without requiring labeled training data.

Algorithm 2 describes the heuristic enhancement process. For a given position transition (before/after move), the module:

(1) Queries Stockfish [10] for objective evaluation metrics
(2) Identifies significant changes in key strategic dimensions

Algorithm 2 Expert-Guided Move Evaluation with Structural Feature Extraction

Require: Position FEN_{before}, Position FEN_{after}
Ensure: Structured expert representation S
$E_{before}, B \leftarrow \text{EvaluatePosition}(FEN_{before})$ ▷ Engine evaluation and best continuation
$E_{after}, M \leftarrow \text{EvaluatePosition}(FEN_{after})$
$\Delta \leftarrow \text{PerspectiveAdjustedDelta}(E_{before}, E_{after})$
$m \leftarrow \text{InferPlayedMove}(FEN_{before}, FEN_{after})$
$F_{white} \leftarrow \text{ExtractPositionFeatures}(FEN_{before}, FEN_{after}, \text{White})$
$F_{black} \leftarrow \text{ExtractPositionFeatures}(FEN_{before}, FEN_{after}, \text{Black})$
$Q \leftarrow \text{AssessMoveQuality}(\Delta, M)$
Construct structured record S with:
  evaluations $(E_{before}, E_{after}, \Delta)$
  played move m and engine-recommended move B
  qualitative assessment Q
  feature deltas (F_{white}, F_{black})
return S

The heuristic extracts a set of interpretable structural feature categories by comparing the board state before and after a move:

- **Evaluation Shift**: Direction and magnitude of the engine-evaluated positional change, adjusted to the perspective of the player who made the move.
- **Material Safety**: Emergence of newly hanging or undefended pieces as a consequence of the move.
- **Pawn Structure**: Introduction of structural weaknesses such as doubled, isolated, or backward pawns.
- **King Safety**: Deterioration of defensive conditions, including loss of castling rights, increased attack pressure, or weakened pawn shields.
- **Positional Control**: Loss of central influence or delayed development of minor pieces.

These features are derived through differential analysis of consecutive positions and do not rely on explicit tactical pattern labeling. Feature categories are implicitly prioritized by the magnitude of the evaluation shift: large negative or positive changes emphasize concrete structural failures or successes (e.g., hanging pieces or king exposure), while minor evaluation changes bias the analysis toward positional factors such as development and center control. This mechanism enables the language model to generate expert-aligned

commentary grounded in objective evaluation signals without requiring supervised concept annotations during training.

3.3 Evaluation Data Construction

While our training corpus contains no position-label pairs, we construct a separate evaluation dataset using the Lichess Elite Database [7] (October 2023) to validate commentary quality. This dataset serves exclusively for assessment, not model training.

We process 700 games from strong players (2400+ Elo) using Algorithm 3. For each position transition, we record:

- **FEN representations**: Before and after the move
- **Engine evaluation**: Centipawn scores and win probabilities from Stockfish 16
- **Move context**: SAN notation and game metadata (optional opening code)

Algorithm 3 Evaluation Dataset Generation from PGN

Require: PGN file P, maximum games N_{max}, Stockfish engine $\mathcal{E}$
Ensure: Evaluation records R

$G \leftarrow \textsc{StreamParsePGN}(P, N_{max})$ ▷ Memory-efficient parsing
$R \leftarrow \emptyset$
for each game $g \in G$ **do**
  $board \leftarrow \textsc{InitialPosition}()$
  $moves \leftarrow \textsc{ExtractMainline}(g)$
  $state_{prev} \leftarrow \textsc{Null}$
  for each move $m \in moves$ **do**
    $board \leftarrow \textsc{Push}(board, m)$
    $(fen, player, eval) \leftarrow \textsc{QueryEngine}(\mathcal{E}, board)$
    if $player$ = Black **then** ▷ Normalize to White perspective
      $win\_prob \leftarrow 100 - \textsc{ConvertToWinProbability}(eval)$
    else
      $win\_prob \leftarrow \textsc{ConvertToWinProbability}(eval)$
    end if
    $state_{curr} \leftarrow (fen, player, win\_prob)$
    if $state_{prev} \neq \textsc{Null}$ **then**
      $r \leftarrow \textsc{CreateRecord}(state_{prev}, state_{curr}, m)$
      $R \leftarrow R \cup \{r\}$
    end if
    $state_{prev} \leftarrow state_{curr}$
  end for
end for
return R

The evaluation dataset schema is detailed in Table 1. Records are stored in CSV format with batched I/O (buffer size 80) for efficiency. All win probabilities are normalized to White's perspective (> 50 favors White) regardless of side-to-move.

Table 1: Evaluation Dataset Schema

Field	Type	Description
`before_fen`	string	FEN before move
`after_fen`	string	FEN after move
`move`	string	Move in SAN format (e.g., `Nf3`)
`after_win_prob`	float	Win probability after move (White perspective, 0–100)
`before_win_prob`	float	Win probability before move (White perspective)
`eval_swing`	float	Centipawn change (negative = Black's move improved)

4 Methodology

Blunder Guard is designed as a lightweight, deployable system for generating instructive chess commentary by combining classical engine evaluation with a Small Language Model (SLM). Rather than relying on large-scale end-to-end model training, the system emphasizes heuristic-guided inference, enabling strong explanatory performance while remaining feasible on consumer hardware. This section outlines the overall system architecture and data flow.

Figure 2 presents an abstract overview of the Blunder Guard pipeline. A user provides a mid-game chess position, which is first converted into a FEN representation and analyzed by the Stockfish engine. The resulting engine evaluation is then processed by a feature extraction module that derives high-level positional concepts. These structured features are passed to a quantized Qwen3 SLM, which generates a natural language explanation describing the impact of the move and the resulting positional changes. The generated explanation is finally returned to the user, completing the inference loop.

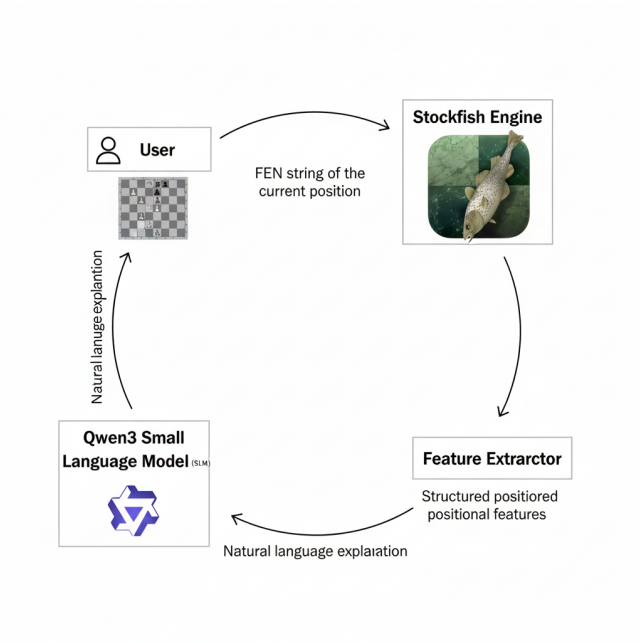

Figure 2: Abstract system architecture of Blunder Guard, illustrating the circular flow between user input, engine evaluation, feature extraction, and language model inference.

4.1 System Overview

The Blunder Guard pipeline follows a modular, sequential design that integrates symbolic chess analysis with neural language generation. Given a chess position and a candidate move, the system produces a natural-language explanation assessing the move's quality and underlying ideas.

The process begins with **engine-based analysis**. A UCI-compatible Stockfish engine evaluates the position before and after the played move, producing centipawn scores, principal variations, and search diagnostics. These signals provide an objective measure of positional change and tactical risk.

Next, a deterministic **feature extraction module** transforms raw engine outputs into high-level conceptual signals. These features include evaluation swing magnitude, material changes, king safety indicators, tactical motifs inferred from engine search depth, and piece activity variations. The extracted features are not used as explicit labels; instead, they guide downstream prioritization during inference.

Based on these features, the system constructs a **heuristic-guided prompt**. The prompt dynamically emphasizes tactical justification in cases of large evaluation swings, while favoring strategic or positional explanations for minor deviations. This mechanism allows the SLM to align its commentary with objective position assessment without requiring explicit concept supervision during training.

Finally, the structured prompt is passed to a fine-tuned **Small Language Model**, which generates human-readable commentary explaining the move in an instructive style. The SLM operates purely at inference time, with no access to engine evaluations beyond what is encoded in the prompt.

This modular architecture decouples chess understanding, heuristic reasoning, and language generation. As a result, Blunder Guard achieves interpretable, pedagogical feedback while maintaining low computational overhead and flexibility for future extensions.

4.2 SLM Fine-Tuning

To adapt the base language model to chess commentary, we perform targeted fine-tuning on our curated chess literature corpus. The fine-tuning procedure emphasizes efficient memory usage and stable training, enabling deployment on consumer-grade GPUs.

4.2.1 Model Architecture. The base model, **Qwen3-1.7B**, is a decoder-only transformer with 28 layers. Each layer contains multi-head attention and feed-forward submodules. Table 2 summarizes the key architectural components:

Input sequences are tokenized and divided into 256-token chunks to optimize training efficiency while preserving context for long-form chess commentary.

4.2.2 QLoRA Adaptation. We employ **QLoRA** [4] (Quantized Low-Rank Adaptation) to efficiently adapt the base model using LoRA adapters. The LoRA configuration uses rank $r = 16$ and scaling factor $\alpha = 32$, applied to selected projection matrices within the attention layers (query and value projections). A small dropout of 0.05 and no bias terms are used to improve generalization and memory efficiency.

The training dataset is prepared as a `Dataset` object and fed to a `Trainer` instance from HuggingFace Transformers. Gradient accumulation is applied to simulate larger batch sizes, and mixed-precision training (FP16) reduces GPU memory usage. After training, the LoRA adapters are merged into the base model, producing a fully adapted model that can be saved and loaded for inference.

This fine-tuning approach balances computational efficiency with performance, enabling the SLM to produce fluent, chess-informed commentary without requiring full-parameter retraining.

4.3 Quantization using `llama.cpp`

To optimize the adapted Qwen3 model for local deployment on limited hardware, we employ **quantization** using the `llama.cpp` toolkit. Quantization reduces model size and memory footprint while maintaining acceptable accuracy for inference. In our workflow, the 16-bit floating-point model is converted to 8-bit using the following command:

```
!llama.cpp/build/bin/llama-quantize \
    /kaggle/working/chess-fp16.gguf \
    /kaggle/working/chess-q8_0.gguf \
    q8_0
```

This produces a compressed model suitable for CPU and GPU inference, enabling efficient evaluation and integration into our Blunder Guard pipeline without sacrificing the quality of generated chess commentary.

4.4 Feature Extraction and Prompting

Our inference pipeline combines domain-specific chess knowledge from Stockfish with the generative capabilities of the fine-tuned LLM. The pipeline is designed to provide human-readable, pedagogical explanations of moves, suitable even for beginner players, while retaining the analytical rigor of engine evaluations.

4.4.1 Positional Feature Extraction. The first step in the pipeline is to extract high-level features from Stockfish output. These features summarize critical aspects of a chess position and serve as structured guidance for the language model. Specifically, the features capture:

- **Central control**: Control over key central squares.
- **Piece activity**: Mobility and coordination of pieces.
- **King safety**: Exposure of the king and potential threats.
- **Pawn structure**: Weaknesses such as isolated or doubled pawns.
- **Initiative**: Who controls the flow of the game and potential tactical threats.

This abstraction allows the model to reason over essential elements of the position without being exposed to raw engine evaluations or complex numerical data.

4.4.2 Inference Approaches. We implement two complementary methods for generating move explanations:

Single Inference Method. In this approach, the LLM directly receives the positional features extracted from Stockfish and generates a natural language explanation in one step. This method is efficient and produces concise commentary, making it suitable for

Table 2: Qwen3-1.7B Model Architecture Overview

Component	Details
Model type	Decoder-only Transformer (Qwen3ForCausalLM)
Embedding layer	Token embedding: 151,936 vocab size, hidden size 2048
Number of layers	28 (Qwen3DecoderLayer)
Attention	Qwen3Attention per layer: - Query projection: 2048→2048 - Key projection: 2048→1024 - Value projection: 2048→1024 - Output projection: 2048→2048 - Layer norms: q_norm, k_norm (RMSNorm, 128-dim)
Feed-forward network (MLP)	Gate: 2048→6144 Up: 2048→6144 Down: 6144→2048 Activation: SiLU
Layer normalization	Input and post-attention RMSNorm (2048-dim)
Positional embedding	Rotary embedding
Language modeling head	Linear: 2048→151,936

fast, real-time feedback or integration into a user interface where simplicity is preferred. The main steps are:

Algorithm 4 Single Inference Method (Abstract)

Require: Stockfish evaluation of current position
Ensure: Beginner-friendly move explanation
1: Extract position features from engine output
2: Construct a natural language prompt using these features
3: Feed the prompt into the LLM
4: Receive and return textual explanation

Chain Inference Method. The chain inference method produces more nuanced explanations by performing multi-step reasoning. First, the LLM evaluates the positions *before* and *after* a move independently, generating structured descriptions for each. These intermediate outputs, together with Stockfish analysis, are then combined in a second prompt to produce a final explanation that clearly articulates why the move improved or weakened the position. This method allows the model to perform its own comparative reasoning in addition to leveraging engine insights, resulting in more detailed and instructive commentary. The workflow can be summarized as:

Algorithm 5 Chain Inference Method (Abstract)

Require: Stockfish analysis of **before** and **after** positions
Ensure: Detailed move explanation
1: Extract positional features for both before and after positions
2: Construct separate LLM prompts for each position
3: Feed prompts into LLM to obtain intermediate textual summaries
4: Compare before/after summaries with Stockfish evaluation
5: Construct a final LLM prompt incorporating all intermediate information
6: Feed final prompt into LLM
7: Receive and return the final textual explanation

4.4.3 Discussion and Use Cases. The single inference method is ideal for scenarios requiring rapid evaluation or lightweight deployment, such as mobile apps or web interfaces. The chain inference method, although computationally heavier, is better suited for detailed post-game analysis, teaching tools, and generating insights that require reasoning over position changes. Together, these two approaches provide a flexible framework that balances speed, clarity, and analytical depth, ensuring the system can serve both casual players and serious learners.

4.5 Deployment and Software Architecture

The Blunder Guard system is implemented as a full-stack web application, combining a Python-based backend for chess analysis with a modern JavaScript frontend for interactive visualization. Unlike many academic or research-oriented projects, Blunder Guard is fully deployable, allowing users to run it locally or on any server, making it accessible across a wide range of devices from desktops to laptops and even mobile browsers. The backend handles model inference, Stockfish evaluation, and API endpoints, while the frontend provides a responsive interface for users to explore game commentary in real time.

4.5.1 Backend. The backend is implemented in the `app` directory using Python and Flask. It is designed for ease of deployment and cross-platform compatibility, and is responsible for:

- Serving API endpoints for LLM inference and Stockfish evaluation
- Managing chess game state and move inputs from the frontend
- Integrating quantized Qwen3 LLM models using `llama.cpp` for efficient local inference on consumer-grade hardware
- Returning structured JSON responses containing position descriptions, move analyses, and suggested explanations
- Supporting seamless deployment in containers, virtual environments, or cloud platforms

4.5.2 Frontend. The frontend is implemented in the `chess-frontend` directory using React and Bun. It is lightweight, performant, and responsive, providing:

- Interactive chessboard display with move highlighting
- Real-time commentary streaming from backend APIs
- User-friendly presentation of positional features, blunder detection, and move suggestions
- Adaptive design for desktop, tablet, and mobile devices, ensuring consistent user experience across platforms
- Simple deployment, allowing the frontend to be served statically or via modern hosting solutions

This architecture ensures smooth communication between the backend LLM inference engine and the frontend visualization, enabling an accessible, deployable, and fully operational chess coaching system that can be run on virtually any device without extensive setup. The software interface is shown in Figure 3.

Figure 3: Blunder Guard user interface showing game analysis panel (left) and interactive chessboard (right).

5 Evaluation

Standard natural language generation metrics such as BLEU, ROUGE, and BERTScore are commonly used to evaluate text generation systems. However, these metrics are not suitable for our setting because **our training corpus contains no paired position–commentary data**. Unlike supervised approaches [5, 6], where models learn to reproduce human commentary for specific positions, Blunder Guard generates *de novo* explanations based on general chess knowledge extracted from literature.

A generated explanation may be linguistically fluent yet strategically incorrect. For example, a commentary stating "White gains a decisive advantage by activating the rook" may read well but be objectively false if the position is evaluated as equal by Stockfish. Conversely, a factually correct explanation phrased in a non-canonical style may appear stylistically unusual while being practically more useful.

5.1 Potential Multi-Dimensional Metrics (Future Work)

While we do not evaluate our system using automated metrics, we note that a comprehensive framework could decouple linguistic quality from strategic accuracy. Possible dimensions include:

- **Strategic Accuracy**: Alignment with engine evaluation, verification of tactical claims, and detection of hallucinated concepts.
- **Pedagogical Utility**: Concept coverage, explanation depth, and actionability for the player.
- **Human Expert Evaluation**: Formal scoring of commentary by rated players using structured rubrics.

These metrics provide a useful reference for future studies, but they were not applied in our current evaluation.

5.2 Expert-Based Assessment

Instead, we performed an informal expert review of model outputs. A chess expert (rated player) examined a representative set of generated commentaries to verify that they were strategically sound and pedagogically meaningful. The overall assessment indicated that the system produces acceptable guidance: explanations were generally accurate and helpful, though not flawless. This approach reflects a realistic evaluation given the absence of gold-standard commentary data.

6 Discussion

Blunder Guard demonstrates that a lightweight, heuristic-guided pipeline combining classical engine evaluation with a Small Language Model (SLM) can generate pedagogically meaningful chess commentary. Unlike fully supervised approaches, our system does not require paired position-commentary datasets, relying instead on strategic knowledge extracted from classical chess literature and structured positional features.

6.1 Strengths of the Approach

Our methodology has several notable advantages:

- **Efficiency and Deployability:** The system can run on consumer-grade hardware thanks to LoRA fine-tuning and quantization with `llama.cpp`, making it more accessible than larger, end-to-end models.
- **Explainability:** By integrating Stockfish evaluation and feature extraction into the pipeline, the model's commentary is anchored in objective analysis, reducing hallucinations common in pure LLM outputs.
- **Flexibility:** The dual inference methods—single-step and chain inference—allow trade-offs between speed and depth of explanation.

6.2 Limitations

Despite these strengths, several limitations should be acknowledged:

- **Qualitative Evaluation:** Our assessment relied on human expert review rather than standardized multi-dimensional metrics. While commentary was generally accurate and informative, we cannot quantitatively measure its strategic alignment across a large dataset.
- **Limited Contextual Depth:** The model operates primarily on immediate positional features and may miss long-term strategic plans that extend beyond the analyzed move.
- **Chess Knowledge Dependency:** The system relies on Stockfish for objective evaluation. In positions where engine assessments are ambiguous or non-intuitive, the commentary may lack subtlety appreciated by human players.

6.3 Future Directions

There are several avenues to improve and extend Blunder Guard:

- **Supervised Data Integration:** Incorporating labeled position-commentary pairs could improve the model's ability to generate accurate and instructive commentary. Such datasets could be created either by expert annotators or by leveraging LLMs to generate synthetic training data.
- **Hardware Optimization:** Optimizing the model to run efficiently on NPUs, mobile devices, or local systems would enhance accessibility and real-time inference capabilities.
- **Expanded Knowledge Base:** Integrating additional chess literature and opening/middle/endgame manuals could improve the breadth and depth of the model's strategic understanding.
- **Robust Evaluation on Seen vs. Unseen Positions:** Splitting chess positions from the training books and evaluating on unseen positions would allow a more systematic assessment of the model's generalization capabilities.
- **Sequence-Level Analysis:** Extending the chain inference method to multi-move sequences could enable the generation of commentary for positional trends and long-term planning.
- **Quantitative Evaluation Framework:** Future work could incorporate metrics such as concept coverage, strategic accuracy, and pedagogical utility to complement human expert review.

Overall, our findings suggest that combining engine evaluation with a fine-tuned SLM provides a promising avenue for accessible, instructive chess commentary systems. Future improvements in training data, hardware optimization, and evaluation strategies can further enhance the model's reliability and practical usefulness.

7 Conclusion

We presented **Blunder Guard**, a lightweight, deployable system for generating pedagogically grounded chess commentary by combining engine evaluation with a fine-tuned Small Language Model (SLM). Our approach leverages unstructured chess literature to teach the model reusable explanatory patterns, while heuristics derived from Stockfish ensure strategic accuracy in generated commentary. Unlike prior supervised or large-scale LLM approaches, Blunder Guard operates efficiently on consumer hardware, enabling fully local deployment and privacy-preserving usage.

Evaluation through expert review indicates that the system produces explanations that are intelligible, relevant, and consistent with engine assessments, even without access to labeled position-commentary datasets. While the results are promising, there remains room for improvement in coverage and stylistic richness. Future work could explore semi-supervised augmentation using labeled or synthetically generated commentary, optimization for NPUs and mobile deployment, and expansion of the training corpus with additional chess literature. Further analysis could also differentiate performance on positions derived from training texts versus unseen positions, providing deeper insight into the model's generalization capabilities.

Acknowledgments

This work was supported by publicly available chess literature, the Stockfish engine, Qwen3 and open-source software such as `llama.cpp`, which enabled local inference and model quantization.

References

[1] David Bronstein. 1979. *Zurich International Chess Tournament, 1953* (revised ed.). Dover Publications, New York, NY. Annotated games from the landmark candidates tournament.

[2] Murray Campbell, A. Joseph Jr. Hoane, and Feng-hsiung Hsu. 2002. Deep Blue. *Artificial Intelligence* 134, 1–2 (2002), 57–83. https://doi.org/10.1016/S0004-3702(01)00129-1 Landmark defeat of world chess champion Garry Kasparov.

[3] Irving Chernev. 1957. *Logical Chess: Move by Move* (1st ed.). Simon & Schuster, New York, NY. Detailed annotations explaining the reasoning behind every move.

[4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. *arXiv preprint arXiv:2305.14314* (2023). https://arxiv.org/abs/2305.14314 Parameter-efficient fine-tuning method used in this work.

[5] Harsh Jhamtani, Varun Gangal, Eduard Hovy, Graham Neubig, and Taylor Berg-Kirkpatrick. 2018. Learning to Generate Move-by-Move Commentary for Chess Games from Large-Scale Social Forum Data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*. Association for Computational Linguistics, Melbourne, Australia, 1661–1671.

[6] Jaechang Kim, Jinmin Goh, Inseok Hwang, Jaewoong Cho, and Jungseul Ok. 2025. Bridging the Gap between Expert and Language Models: Concept-guided Chess Commentary Generation and Evaluation. *arXiv preprint arXiv:2410.20811* (2025). arXiv:2410.20811 [cs.LG]

[7] Lichess.org. 2023. Lichess Elite Database. https://database.nikonoel.fr/ High-quality chess games from strong human players (2400+ Elo).

[8] Jeremy Silman. 2010. *How to Reassess Your Chess: Chess Mastery Through Chess Imbalances* (4th ed.). Siles Press, Los Angeles, CA. Comprehensive guide to positional play and strategic thinking.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2018. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play. *Science* 362, 6419 (2018), 1140–1144. https://doi.org/10.1126/science.aar6404

[10] Stockfish Team. 2024. Stockfish: A Free and Strong Chess Engine. https://stockfishchess.org/ Open-source chess engine used for position evaluation.

[11] Mikhail Tal. 1997. *The Life and Games of Mikhail Tal* (revised ed.). Everyman Chess, London, UK. Autobiography and annotated games of the eighth World Chess Champion.

[12] Siegbert Tarrasch. 1935. *The Game of Chess* (1st ed.). David McKay Company, Philadelphia, PA. Classic instructional text covering fundamental strategic principles.