CHESSQA: EVALUATING LARGE LANGUAGE MODELS FOR CHESS UNDERSTANDING

Qianfeng Wen, Zhenwei Tang, and Ashton Anderson
Department of Computer Science
University of Toronto
qianfeng.wen@mail.utoronto.ca, {josephtang,ashton}@cs.toronto.edu

ABSTRACT

Chess provides an ideal testbed for evaluating the reasoning, modeling, and abstraction capabilities of large language models (LLMs), as it has well-defined structure and objective ground truth while admitting a wide spectrum of skill levels. However, existing evaluations of LLM ability in chess are ad hoc and narrow in scope, making it difficult to accurately measure LLM chess understanding and how it varies with scale, post-training methodologies, or architecture choices. We present ChessQA, a comprehensive benchmark that assesses LLM chess understanding across five task categories (Structural, Motifs, Short Tactics, Position Judgment, and Semantic), which approximately correspond to the ascending abstractions that players master as they accumulate chess knowledge, from understanding basic rules and learning tactical motifs to correctly calculating tactics, evaluating positions, and semantically describing high-level concepts. In this way, ChessQA captures a more comprehensive picture of chess ability and understanding, going significantly beyond the simple move quality evaluations done previously, and offers a controlled, consistent setting for diagnosis and comparison. Furthermore, ChessQA is inherently dynamic, with prompts, answer keys, and construction scripts that can evolve as models improve. Evaluating a range of contemporary LLMs, we find persistent weaknesses across all five categories and provide results and error analyses by category. We will release the *code*, periodically refreshed *datasets*, and a public leaderboard to support further research.

1 INTRODUCTION

Since even before the term "artificial intelligence" was coined, chess has been a central yardstick for measuring our progress towards building machines that think. In the 1950s, Shannon and Turing wrote seminal papers on how computers might be programmed to play chess, and in the decades that followed researchers tracked how artificial intelligence was progressing by pitting computer chess programs against increasingly strong human opponents (Shannon, 1950; Turing, 1953). IBM's Deep Blue defeated the reigning World Champion, Garry Kasparov, in 1997, ushering in a period where bespoke models achieved superhuman ability in chess and measuring AI progress with chess was less common (Campbell et al., 2002). But with the rise of generative artificial intelligence and large language models (LLMs), chess has returned as a useful barometer for assessing the depth and quality of machine thinking. How good is the latest LLM, and how good is its reasoning? In a perhaps surprising return to decades-old practice, we are using chess to find out.

Assessing LLM capabilities is essential to AI progress. Generative AI is rapidly transforming industries, the economy, and life in general by providing access to flexible and general intelligence that is broadly useful across domains and needs. The trajectory of this transformation is largely guided by model capability, thus accurately measuring it can help us understand the dynamics of the fundamental shifts taking place. Furthermore, it is equally important to understand model limitations, to support appropriate reliance on LLM outputs and to suggest directions for future improvement. Chess has several desirable properties as a domain for assessing LLM capabilities. It is a neatly circumscribed, perfect-information, and fully objective domain with clear states, actions, goals, and rewards. Despite this, it is incredibly deep, and admits a huge variety of skill levels. People are also still very active in playing chess, and a near-infinite supply of human chess data is available on-

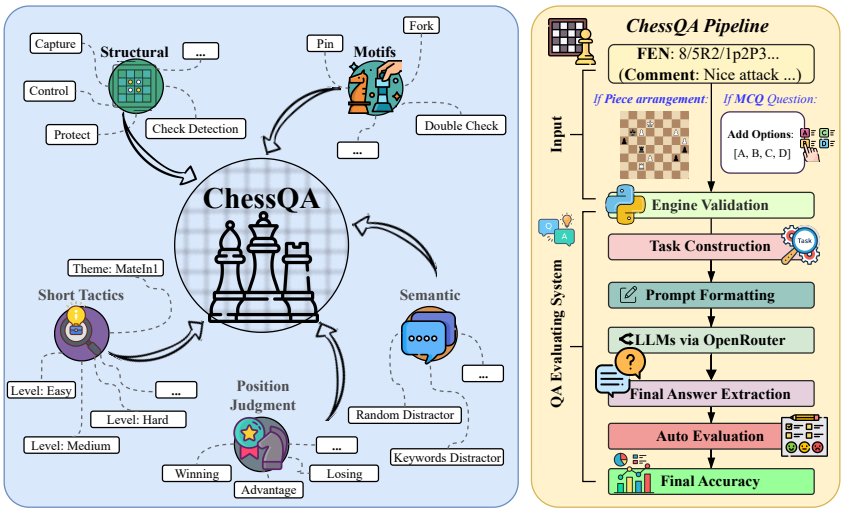


Figure 1: ChessQA at a glance.

line. Importantly, chess remains anecdotally difficult for LLMs, with contemporary state-of-the-art models routinely failing to complete even simple chess tasks.

However, using chess to assess LLM capabilities is difficult. The sheer vastness of chess means it is not straightforward to actually measure how LLMs do across the entire domain. Most existing ways of using chess to assess models are ad hoc and narrow, often only measuring move quality, performance, or perhaps a narrow set of concepts.

We introduce ChessQA, a unified, comprehensive benchmark that assesses LLM chess understanding across five increasingly complex categories, covering approximately the increasingly abstract levels of understanding that players master as they accumulate chess knowledge. These task categories are Structural (basic rules), Motifs (recurring patterns), Short Tactics (short, targeted calculations), Position Judgment (position evaluation), and Semantic (describing essential high-level concepts in a given position). By constructing a benchmark this way, we capture a more comprehensive picture of chess ability and understanding, going significantly beyond the simple move quality evaluations done previously, and offer a controlled, consistent setting for diagnosis and comparison. Furthermore, ChessQA is inherently dynamic, with prompts, tasks, and construction methods that can evolve as models improve. Evaluating a range of contemporary LLMs, we find persistent weaknesses across all five categories and provide results and error analyses by category.

2 Related Work

Frontier LLMs. Early general-purpose LLMs scaled decoder-only transformers and were then instruction-tuned and aligned for daily conversations. Contemporary LLMs further scale the model, enable longer context windows, integrate agentic and multi-modal capabilities, and use reinforcement learning for better alignment. Representative proprietary model series include GPT (OpenAI, 2024c; 2025c;d;a; 2024a;b; 2025b), Claude (Anthropic, 2024; 2025a;b), Gemini (DeepMind, 2023; 2024a;b; Comanici et al., 2025), and DeepSeek (DeepSeek-AI, 2024). Popular open-source models include LLaMA series (Grattafiori et al., 2024; Touvron et al., 2023), Qwen series (Yang et al., 2025), Mistral models (team, 2024), and Gemma (Team et al., 2024; 2025).

LLM evaluation via QA and reasoning. Question answering (QA) benchmarks have evolved from extractive span selection on short passages (SQuAD; SQuAD 2.0) to open-domain pipelines that learn to retrieve and generate (MS MARCO, Natural Questions, ORQA, REALM, TravelDest) (Rajpurkar et al., 2016; 2018; Bajaj et al., 2016; Kwiatkowski et al., 2019; Lee et al., 2019; Guu et al., 2020; Wen et al., 2024; 2025). To probe multi-step reasoning and robustness, compositional and domain QA introduced multi-hop aggregation, distractors, and truthfulness (HotpotQA, MultiRC, MuSiQue, AmbigQA; ARC, QASC, OpenBookQA, SciQ, CommonsenseQA, BoolQ;

Table 1: **Positioning of *ChessQA*.** Prior work typically targets a single facet (move selection, state tracking, commentary, or gameplay). *ChessQA* unifies five categories with verifiable scoring: *Structural*, *Motifs*, *Short Tactics*, *Position Judgment*, and *Semantic*.

	Primary focus	**Input → Output**	**Primary metric(s)**	**Response type**	**ChessQA mapping**
MATE (Wang et al., 2024a)	Move selection with strategy/tactic rationales	FEN + candidate moves → best move (+ explanation)	Top-1 over candidates; explanation quality	MCQ + free-form rationale	*Short Tactics*
PGN2FEN (Cooper, 2025)	State tracking	PGN/SAN sequence → FEN	Exact-FEN; legality gating	Free-form FEN string	*Structural*
LLM Chess Puzzles (Kagi Search, 2025)	Single-move best-move accuracy	FEN → SAN move	Top-1 accuracy (by difficulty)	Free-form SAN	*Short Tactics*
LLM Chess Leaderboard (Saplin, 2025)	Text-only gameplay strength	Dialogue (text) moves → full game vs. engine	Elo vs. calibrated engine; win rate (%)	Open-ended dialogue (agent play)	*Short Tactics*
Concept-guided Chess Commentary (Kim et al., 2024)	Commentary generation & evaluation	FEN + text → commentary / score	Task-specific human/automatic protocol	MCQ or scored free-form commentary	*Semantic*
Kaggle Game Arena (Lee et al., 2025)	Head-to-head *game* evaluation	Game API/dialogue → complete games	Game Outcome	Chess Moves	*Short Tactics*; *Position Judgment*

DROP; TruthfulQA) (Yang et al., 2018; Khashabi et al., 2018; Trivedi et al., 2022; Min et al., 2020; Clark et al., 2018; Khot et al., 2020; Mihaylov et al., 2018; Johannes Welbl, 2017; Talmor et al., 2018; Clark et al., 2019; Dua et al., 2019; Lin et al., 2021). As general LLMs saturated earlier sets, exam-style and expert-curated evaluations (MMLU, MMLU-Pro, GPQA, AGIEval) and math benchmarks (GSM8K, MATH, OlympiadBench, Omni-MATH) emphasized harder, diagnostic items (Hendrycks et al., 2020; Wang et al., 2024b; Rein et al., 2024; Zhong et al., 2023; Cobbe et al., 2021; Hendrycks et al., 2021; He et al., 2024; Gao et al., 2024). Long-context stress tests (QuALITY, QASPER, LongBench, RULER) target scaling limits in context use (Pang et al., 2021; Dasigi et al., 2021; Bai et al., 2023; Hsieh et al., 2024).

Chess AI. Modern chess engines couple deep evaluation with powerful search (AlphaZero; Leela Chess Zero; NNUE-augmented Stockfish) (Silver et al., 2017; 2018; Leela Chess Zero Project, 2025; Stockfish Developers, 2020b; 2025; Nasu, 2018). Parallel works model *human* decision making in chess (Maia; Maia-2; Maia4All) (McIlroy-Young et al., 2020; Tang et al., 2024; 2025a). On the language side, prior efforts curate commentary corpora and study generating or scoring analyses, often with engine signals in the loop (Jhamtani et al., 2018; Zang et al., 2019; Lee et al., 2022; Feng et al., 2023; Kim et al., 2024).

Chess-specific LLM evaluations. Recent efforts evaluate language models on narrow chess subskills. **Move-selection** corpora such as MATE curate large position sets with expert-annotated strategic/tactical rationales and report fine-tuned models outperforming proprietary baselines on *select-the-best-move* tasks (Wang et al., 2024a). **State-tracking** benchmarks like PGN2FEN stress whether models can convert move sequences into exact board states, highlighting long-horizon legality and reconstruction failures (Cooper, 2025). Community **puzzle/best-move** leaderboards evaluate Top-1 accuracy on single-move positions (Kagi Search, 2025), while **text-only gameplay** leaderboards estimate Elo, the standard chess rating system that quantifies player strength (Elo, 1978), by playing against calibrated engine levels (Saplin, 2025). Beyond move choice, **commentary evaluation** benchmarks introduce concept-guided generation and scoring protocols that align textual analysis with engine signals (Kim et al., 2024). We also note the popularity of the Kaggle Game Arena, which provides large-scale head-to-head *gameplay* leaderboards and serves as a complementary, agentic lens rather than a QA benchmark (Lee et al., 2025). Table 1 situates representative efforts alongside ours. Orthogonally, **cross-modal isomorphic** evaluations such as SEAM and ISOBENCH include chess tasks (e.g., FEN strings vs. board images) to test representation-invariant reasoning and consistently reveal modality-specific performance gaps even when semantics are held constant (Tang et al., 2025b; Fu et al., 2024).

3 CHESSQA

We introduce **ChessQA**, a 50-task, 3,500-item benchmark that systematically probes LLM chess understanding along a curriculum of five categories: 11 *Structural* tasks, 6 *Motifs* tasks, 24 *Short Tactics* tasks, 5 *Position Judgment* tasks, and 4 *Semantic* tasks (the distribution of tasks is shown in Figure 2). Going beyond ad hoc testing of move quality or narrow semantic categories, ChessQA comprehensively includes rules knowledge, pattern recognition, concrete calculation, evaluative judgment, and high-level semantics in a controlled setting. In what follows, we outline the benchmark's design principles, specify the prerequisites, and detail breakdown of each category.

3.1 DESIGN PRINCIPLES

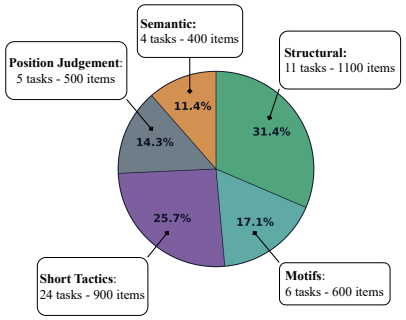


Figure 2: Task distribution in ChessQA.

Comprehensive dimension coverage. Ideally, a benchmark that measures chess understanding in LLMs should test two fundamental dimensions. First, it should rigorously measure chess concepts, it must reflect what players actually use over the board: understanding rules and concrete board states; recognizing recurring patterns and motifs; carrying out short, local calculations; judging positional factors; and articulating clear, instructive explanations. (ii) Comprehensive LLM ability evaluation, it must exercise comprehensive reasoning: from fast, short-term checks (legality, immediate tactics, etc.) to slower, long-term judgment like forecasting the trajectory of advantage via bucketed centipawn outcomes and selecting the most informative commentary among plausible alternatives. With this target in mind, **ChessQA** instantiates both dimensions: we sample openings, middlegames, and endgames; include both sides to move; and balance tactical and positional positions. Each item is crafted either to elicit quick calculation or to demand sustained evaluation and explanation, so measured performance tracks genuine chess understanding rather than recall of a few memorized patterns.

Spectrum of abstraction levels. Mastering chess, like other subjects, involves understanding concepts and developing knowledge at progressively deeper and more complex levels of abstraction. We design tasks that span five increasingly complex categories that mirror how people typically learn chess: first they learn the rules, then recognize patterns, then calculate short lines, then judge positions, and finally explain ideas. Concretely, *Structural* tests for board awareness and single-step legality; *Motifs* requires naming or spotting a chess motif that doesn't require long calculation; *Short Tactics* pushes tests the model's ability to solve tactical puzzles derived from real games; *Position Judgment* asks for an overall quantitative assessment that approximates a chess engine's evaluation; and *Semantic* requires the model to select the most relevant and insightful natural-language comment describing the position from a set of possible comments. This comprises a natural progression in understanding and ability from recognition to reasoning to explanation.

Calibrated task difficulty. Ideally, a chess benchmark should have calibrated, controllable difficulty so that we can test models at various levels of complexity. We use two principled levers to achieve this. **(i) Data selection:** we raise difficulty by sourcing tougher positions, e.g., in our *Short Tactics*, we can choose higher-rated tactical puzzles (that have earned their high ratings by being hard to solve in online puzzle solving against humans) so the underlying chess task is more difficult without changing the task format. **(ii) Option design:** with the position fixed, we can make the answer choice set more demanding. In our *Position Judgment* tasks, we can tighten centipawn granularity by shrinking the bucket grid from a coarse $\{-400, -200, 0, 200, 400\}$ to a finer $\{-200, -100, 0, 100, 200\}$, to modulate the difficulty of selecting the correct answer. Also in our *Semantic* tasks, we control difficulty by varying the method by which we select "distractor" incorrect multiple-choice answers via retrieval: we find comments that are closer to the truth yet objectively wrong, forcing deeper understanding. All of these knobs are parameterized and versioned so that our benchmark scores will remain comparable across future releases as the benchmark keeps pace with rapidly improving LLMs.

3.2 PREREQUISITES

Notations. Chess is equipped with standardized textual representations: board positions are written in Forsyth–Edwards Notation (FEN), complete games in Portable Game Notation (PGN), and individual moves in either Universal Chess Interface (UCI) or Standard Algebraic Notation (SAN). We use only these notations in prompts and answers. For moves, we choose UCI over SAN because UCI is deterministic at the string level: the source square, target square, and (if any) promotion piece uniquely specify the move and do not rely on board state for disambiguation. This avoids SAN's position-dependent variants (e.g., file/rank disambiguators and check/mate suffixes) and makes parsing and exact-match scoring straightforward. Since FEN/PGN/UCI are ubiquitous in online databases, engines, and forums, models have almost certainly seen them during pretraining and instruction tuning. Using these standards enables a fair zero-shot evaluation without introducing ad hoc formats. (Edwards, 1994; Wikipedia contributors, 2025; Fiekas, 2024a; python chess, 2024)

Data Sources. We use data sources that cover broad chess knowledge and provide reliable, objective targets. The Lichess Puzzles corpus (Team, 2025b) is a massive set of puzzles that people continuously train on, and each puzzle is annotated with an empirically-derived rating describing its difficulty and theme tags that describe the motifs involved in the puzzle. The Lichess Evaluations release (Team, 2025a) is a set of aggregated Stockfish analyses for millions of positions at varying depths, supplying centipawn targets and principal variations for each position. For natural-language understanding, we draw expert commentary from ChessBase 17 (ChessBase GmbH, 2023), a human-annotated game database that provides semantically rich explanations.

Tools. We pair reliable chess tooling with scalable retrieval to construct the benchmark. We use `python-chess` (Fiekas, 2024a) to parse and validate FEN/PGN, enforce move legality, normalize all moves to UCI, and orchestrate engine calls. Stockfish (Stockfish Developers, 2020a; 2025) supplies ground-truth centipawn evaluations and principal variations at controlled depths. For retrieval-based distractors in *Semantic*, we index candidate commentary with FAISS (Johnson et al., 2017) and dense embeddings, allowing us to retrieve semantically close yet incorrect options and to increase difficulty at will.

3.3 BENCHMARK CONSTRUCTION

***Structural*: Basic chess rule understanding.** The first category (11 tasks, 1100 items) measures basic rule-level competence, such as recognizing legal moves, checks, attacked/controlled/protected squares, piece locations, and deterministic board state updates that don't require search or in-depth evaluation. We construct 11 tasks with data derived from public Lichess positions and games, and categorize them into three subtasks: board-state recognition (read what is currently on the board), single-ply[1] (select or validate one legal move), and state updates (apply move sequences and output the board state corresponding to the exact resulting position). Details are shown in Table 2. We also enforce simple legality constraints in the generators: e.g., a piece pinned to its King cannot control or protect squares, and "protect" excludes the king as a target. More details about task construction and the pesudocode are presented in Appendix C.1.

Table 2: ***Structural*** tasks overview.

Subtask	Input	Output
Board-State Recognition		
Piece arrangement	FEN	Piece list
Check detection	FEN	Pieces on square
Capture squares	FEN + piece on square	Squares
Control squares	FEN + piece on square	Squares
Protect squares	FEN + piece on square	Squares
Single-Ply		
Legal moves (piece)	FEN + target square	UCI moves
Legal moves (all)	FEN	UCI moves
Check-in-1	FEN	UCI moves
State Updates		
State tracking—short	start FEN + 1–5 UCI	FEN
State tracking—mid	start FEN + 6–10 UCI	FEN
State tracking—long	start FEN + 11–15 UCI	FEN

***Motifs*: Chess motifs recognition.** Beyond basic rules, the next step on the path to chess fluency is recognizing tactical motifs and the forcing moves that create them. The *Motifs* category of ChessQA includes 6 tasks and 600 items, which divides motifs into two sub-categories: (1) static motifs (pins, skewers, batteries, and fork squares), which can be identified directly from board position without

[1] In chess, a *ply* is one half-move, i.e. one move made by a single side, either White or Black. A full move is two plies: one half-move by White and one half-move by Black.

move simulation, and (2) dynamic motifs, which emerge only through one-ply lookahead search. We input FEN notation of chess positions for all the tasks in this category, and the expected outputs are detailed in Table 3.

***Short Tactics*: Chess tactic implementation.** Once a model can handle rules and common motifs, the next step is to apply tactics to find the best move in positions. While regular game positions provide diverse data, chess puzzles form a carefully crafted subset of regular positions with a unique best move that typically leads to a decisive advantage. Such a property of chess puzzles enables us to find an absolute best move as ground truth for a given position, instead of distinguishing between multiple moves of similar quality. Furthermore, puzzles are specifically designed to isolate and teach important tactical patterns and strategic concepts, which are particularly suitable for evaluating an LLM's ability to execute short tactics. Therefore, we choose to use chess puzzles from Lichess Database [2] to construct samples for this category. This category contains 24 subtasks and 900 items, organized according to empirical difficulty recorded by Lichess and the themes of the tactics involved. We measure the first move correctness instead of requiring the LLMs to finish predicting the entire principal variation. We only include puzzles for which the ground truth principal variations are under 6 ply, so as to keep the planning required for LLMs relatively short. Details about subtasks included are reported in Appendix C.3.

Table 3: ***Motifs*** subtasks overview

Subtask	Output
Pins	`pinning>pinned>target`
Skewers	`attacker>front>back`
Forks	`forking>sq1-sq2(-sq3...)`
Batteries	`square>square(>square...)`
Discovered checks	UCI moves
Double checks	UCI moves

***Position Judgment*: Long-term evaluation.** Once an agent has achieved fluency in how to combine motifs into tactical sequences, the next level of chess understanding is evaluating an entire position. To accurately predict an engine's centipawn advantage (cp) evaluation of a position requires long-horizon strategic reasoning and multi-step planning, making it an ideal task category in our ChessQA benchmark for assessing models' capacity for extended chain-of-thought reasoning and long-term positional understanding in chess. Specifically, centipawn advantage is a chess evaluation metric that quantifies positional superiority in hundredths of a pawn, where +100 represents a material or positional advantage equivalent to being one pawn ahead for White. Questions in this category ask the model to select Stockfish centipawn evaluation in the current position from a set of possible options. For each position s, we take the deepest available engine evaluation record from Lichess Evals [3] to determine the ground truth centipawn advantage of s, excluding positions where checkmate can be forced for either side. We construct 5 evaluation subtasks by sampling 100 positions for each of five distinct centipawn advantage ranges: Losing (-400±50 cp), Disadvantageous (-200±50 cp), Neutral (0±50 cp), Advantageous (200±50 cp), and Winning (400±50 cp), all evaluated from White's perspective, yielding 500 instances across the full spectrum of positional evaluation. We always give the options as $\{-400, -200, 0, 200, 400\}$ for LLMs to select from. Unlike previous categories that are testing chess understanding with isolated static features or short dynamics, centipawn evaluation requires long-term look-ahead and integrating multiple positional dimensions into a holistic assessment that reflects both immediate tactics and strategic planning.

***Semantic*: Chess-specific language understanding.** Beyond choosing moves and making judgments, arguably the highest abstraction level of chess mastery is the ability to talk fluently about a position, and explain one's reasoning. A capable model should be able to connect a concrete move and position to the strategic and tactical ideas at play. This category (4 tasks, 400 items) evaluates grounded natural-language understanding tied to a specific move and position. Each item provides the FEN, the just-played UCI move; four human comments are shown, exactly one of which is true, specific, and informative for that context. Distractors create lexical, structural, and semantic confounds. The model answers with a single letter (`A/B/C/D`). Comment text is anonymized and normalized (e.g., player names are replaced with the color they are playing); a strict position-truth judge admits only comments whose claims are supported by the paired move and position. The *random* subtask serves as a baseline with i.i.d. sampled distractors, while three confound-controlled subtasks gradually introduce more difficulty: *keyword* introduces lexical confusion by selecting distractors sharing chess terminology (e.g., fork, safety), *piece_stage* creates structural ambiguity using

[2] https://database.lichess.org/#puzzles

[3] https://database.lichess.org/#evals

comments from positions with identical piece types and game phases, and *embedding* finds distractors through dense retrieval of similar comments using Qwen3-Embedding-8B (Zhang et al., 2025). This progression from random to semantically similar distractors enables fine-grained assessment of models' chess-specific language understanding beyond pattern matching. More details are reported in Appendix C.

4 EXPERIMENTS

We conduct extensive experiments to evaluate the chess understanding capabilities of frontier LLMs on our proposed ChessQA benchmark. We aim to probe the general and per-category performance of LLMs as well as their token- and cost-effectiveness.

4.1 EXPERIMENTAL SETTINGS

We evaluate 15 contemporary LLMs spanning 7 families on ChessQA, including Anthropic, DeepSeek, Google, Meta, Mistral AI, OpenAI, and Qwen. Models are treated strictly as black-box text-to-text models without agentic behavior, such as tool use, online search, and code execution. We enable thinking with medium efforts if the model supports thinking mode, allowing at most 32K tokens to be generated. Other non-thinking models are given 8,192 tokens as the budget to generate responses. We ran GPT-5, Gemini 2.5 Pro/Flash, Claude Sonnet 4, DeepSeek V3.1, and Qwen3 Next 80B with both thinking and non-thinking modes to investigate the effectiveness of generating long Chain-of-Thought reasoning chains for chess understanding, resulting in 23 runs in total. We run all models under their default sampling parameters for generation using OpenRouter (OpenRouter, 2023). We extract the final answer with pre-defined patterns that have been added to the prompts and use exact matching to evaluate the correctness of each response. Specifically, for questions that require multiple answers (e.g., find all legal moves), we first parse the extracted final answer into a set, and then apply exact set matching to evaluate its correctness. This design avoids extraneous errors introduced in answer processing, enabling us to better isolate and measure the chess understanding capabilities of LLMs.

4.2 RESULTS

Overall performance. As shown in Fig 3, most models achieve relatively low performance on our ChessQA benchmark, with only 4 runs achieving over 50%. GPT-5* with thinking is the best performing model, achieving an accuracy of 79.3%. Unlike in many open benchmarks (Wang et al., 2024b) where proprietary models significantly outperform open-source alternatives in general, we found state-of-the-art open-source models such as DeepSeek v3.1* and Qwen3 Next 80B* ranking high in our leaderboard. Performance consistently follows scaling laws across both proprietary and open-source model families, with larger variants outperforming their smaller counterparts (Gemini-2.5-Pro* at 43% vs. Gemini-2.5-Flash* at 29%; Llama 4 Maverick at 17% vs. Scout at 9%), highlighting the impact of model scale on chess understanding capabilities.

Performance across categories. As shown in Fig 3, model performance differs significantly across categories, even the best performing GPT-5* with thinking that can achieve 97% in the empirically easiest *Structural* tasks, can only achieve 40% in *Position Judgment*. Notably, *Short Tactics* represent the hardest category (mean 17.4%) and most models fall under 20%. Also, *Position Judgment* accuracy remains stubbornly limited even among top-performing models. Since *Position Judgment* are essentially 5-way classification problems, a random guess will theoretically get 20%. These two categories, which involves short and long-term look-ahead in chess, appear to be more challenging for LLMs than other static pattern recognition categories, demonstrating their deficiency in implicit search and long-term planning.

Reasoning efforts and token efficiency. Our analysis highlights the substantial impact of reasoning efforts: models employing explicit reasoning consistently outperform their base counterparts. Pairwise comparisons indicate an average accuracy improvement of **+14.7** percentage points when reasoning is activated (e.g., GPT-5 improved from 44.0% to 79.3%, DeepSeek v3.1 from 39.6% to 56.4%, and Claude Sonnet 4 from 41.7% to 51.8%). From Fig 5, we analyzed token utilization across all models, observing significant variability. The number of tokens employed per task

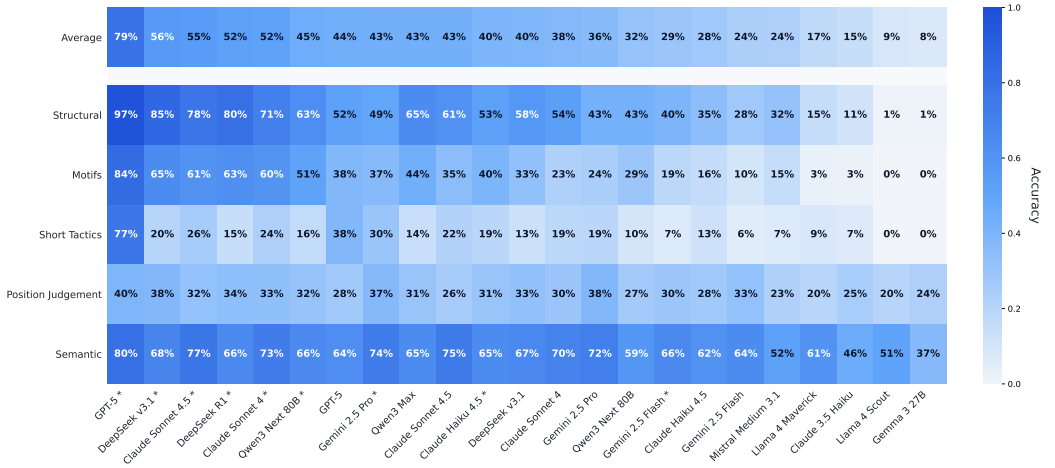


Figure 3: The overall and per-category performance comparison. * denotes thinking enabled.

Figure 4: Breakdown of response evaluation results.

varied significantly, ranging from approximately 497 tokens to over 13,452 tokens. Models using explicit reasoning strategies consistently consumed more tokens (9,142 tokens on average for GPT-5*) compared to their non-reasoning counterparts (1,236.31 tokens on average for GPT-5). Notably, as shown in Appendix Fig 7, higher token usage correlated positively with accuracy improvements across nearly all categories, suggesting the effectiveness of test-time scaling. However, we question the token efficiency of models as human chess players won't need an average of 11,668 tokens per problem to fully verbalize the solution to a chess puzzle. Therefore, the token efficiency is an important direction towards better LLMs, in particular for chess understanding.

Error analysis. We examined incorrect GPT-5* responses and identified four frequent failure modes: (1) board-state hallucination or incorrect piece/square recognition, (2) legality reasoning errors in short tactical problems, (3) correct analytical reasoning leading to an incorrect final move choice, and (4) false assertions of "no answer." In mode (1), GPT-5* occasionally misreads board states, such as confusing the knight's position or inventing phantom pieces, leading to illegal moves (Appendix D.1). Mode (2) revealed contradictions or flawed heuristics about move legality (e.g.,

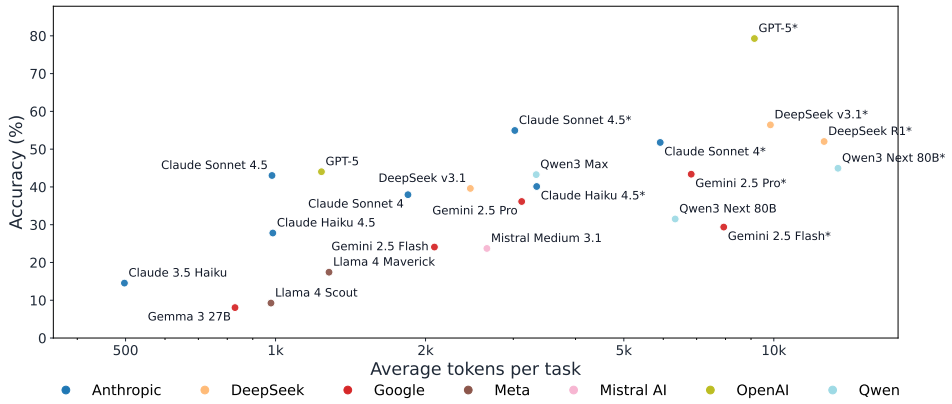


Figure 5: Performance comparison w.r.t #tokens per problem. * denotes thinking enabled.

forced move contradictions or misapplication of adjacency heuristics; Appendix D.2). Mode (3) demonstrated instances where sound intermediate reasoning resulted in suboptimal final moves, like overlooking underpromotion advantages or selecting inferior tactical continuations (Appendix D.3). Lastly, mode (4) captures instances where GPT-5* erroneously concluded "no answer," neglecting existing valid solutions due to misapplication of tactics or misinterpretation of positional constraints (Appendix D.4). To verify the existence of the board-state hallucination problem, we explicitly add piece arrangement context into the prompts. As shown in Table 4 in the Appendix, when piece arrangements are given, the overall performance is significantly improved. Such a setting is equivalent to eliminating the difficulty of board-state recognition for LLMs, reflecting the severity of the board-state hallucination problem in LLM reasoning.

5 Discussion

Chess as measurement tool. Chess was called the "drosophila of AI" by John McCarthy and others as artificial intelligence was first developed. Bespoke, domain-specific programs eventually surpassed human ability in chess, but with the advent of general artificial intelligence, chess has resumed its role as the ideal testbed for measuring our progress towards developing AI. Chess is perfectly objective with clear states, actions, and rewards, it spans a deep skill gradient with abundant human data, and yet modern LLMs still fail at seemingly simple chess tasks. Existing chess-LLM evaluations are typically ad hoc (best-move, narrow tasks, gameplay), which makes holistic measurement difficult. Our contribution is to unify these perspectives in a controlled QA setting that measures broad understanding and capabilities.

Takeaways. Across models and categories we find persistent weaknesses, but explicit reasoning consistently helps: enabling it yields sizeable average gains (+14.7 points on pairwise comparisons), with higher token budgets correlating with higher accuracy, suggesting models effectively leverage additional reasoning tokens. The benchmark is robust to prompt-format perturbations, and adding piece-arrangement context shifts category accuracies in predictable ways. Our qualitative error analysis reveals four recurrent failure families: (1) board-state hallucination/misrecognition; (2) legality mistakes in short tactics; (3) sound analysis but wrong final action; and (4) false "no answer." These patterns indicate that state parsing and action selection remain bottlenecks even when intermediate analysis is plausible.

Future work. By spanning rules, patterns, calculation, evaluative judgment, and explanation in a single, verifiable QA framework, ChessQA provides a diagnostic map for intervention. Because the pipeline is parameterized and versioned, and because chess is practically infinite with respect to data, we can periodically refresh evaluation data to maintain perennially-fresh leaderboards, as well as steadily increase difficulty and maintain comparability as models improve. Together, these aspects make ChessQA a sustained, useful measurement suite for LLM chess understanding.

6 ETHICS STATEMENT

This work adheres to the ICLR Code of Ethics. No human subjects or animal experimentation were involved in this study. All datasets utilized, including Lichess puzzle databases and public chess game archives, were obtained in compliance with relevant usage guidelines, ensuring no violation of privacy. We have made concerted efforts to prevent biases and discriminatory outcomes throughout our research process. No personally identifiable information was utilized, nor were any experiments conducted that could compromise privacy or security. We are committed to upholding transparency and integrity throughout the research process.

7 REPRODUCIBILITY STATEMENT

We have made significant efforts to ensure the reproducibility of the results presented in this paper. All code and datasets are publicly available through an anonymous repository, enabling replication and verification of our experiments. The experimental setup, including model evaluations, prompt construction, and hardware specifics, is comprehensively detailed within the paper. Additionally, we have provided a thorough description of our contributions, including the design and construction of the ChessQA benchmark, generation and curation of datasets spanning Structural, Motifs, Short Tactics, Position Judgment, and Semantic categories, implementation of evaluation scripts, and extensive analyses of contemporary LLM performance.

Furthermore, public datasets utilized in this work, such as Lichess puzzle sets and FEN/PGN archives from chess databases, are openly accessible, ensuring consistent and reproducible evaluation outcomes. We believe these measures will facilitate the reproduction of our findings and contribute to ongoing advancements in this field.

REFERENCES

Anthropic. The claude 3 model family: Opus, sonnet, haiku, 2024. URL https://assets.anthropic.com/m/61e7d27f8c8f5919/original/Claude-3-Model-Card.pdf.

Anthropic. Claude 3.7 sonnet system card, 2025a. URL https://anthropic.com/claude-3-7-sonnet-system-card.

Anthropic. System card: Claude opus 4 & claude sonnet 4, 2025b. URL https://www-cdn.anthropic.com/4263b940cabb546aa0e3283f35b686f4f3b2ff47.pdf.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artif. Intell.*, 134(1–2): 57–83, January 2002. ISSN 0004-3702. doi: 10.1016/S0004-3702(01)00129-1. URL https://doi.org/10.1016/S0004-3702(01)00129-1.

ChessBase GmbH. Chessbase 17: Commenting functions (text & variants). https://help.chessbase.com/cbase/17/eng/commenting_functions.htm, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Aidan Cooper. Pgn2fen: A benchmark for evaluating llm chess reasoning. Blog post, May 2025. URL https://www.aidancooper.co.uk/pgn2fen-benchmark/. Accessed 2025-09-16.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021.

Google DeepMind. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Google DeepMind. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024a.

Google DeepMind. Introducing gemini 2.0: our new ai model for the agentic era, 2024b. URL https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024/.

DeepSeek-AI. Deepseek-v3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

Steven J. Edwards. Portable game notation (pgn) specification and implementation guide. https://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm, 1994.

Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. Chessgpt: Bridging policy learning and language modeling. *Advances in Neural Information Processing Systems*, 36:7216–7262, 2023.

Niklas Fiekas. python-chess: a chess library for python. https://python-chess.readthedocs.io/, 2024a. Move generation/validation, FEN/PGN/SAN/UCI support, pins/checks.

Niklas Fiekas. python-chess: Engine (uci/xboard) interface. https://python-chess.readthedocs.io/en/latest/engine.html, 2024b.

Deqing Fu, Ruohao Guo, Ghazal Khalighinejad, Ollie Liu, Bhuwan Dhingra, Dani Yogatama, Robin Jia, and Willie Neiswanger. Isobench: Benchmarking multimodal foundation models on isomorphic representations. *arXiv preprint arXiv:2404.01266*, 2024.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pp. 3929–3938. PMLR, 2020.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Harsh Jhamtani, Varun Gangal, Eduard Hovy, Graham Neubig, and Taylor Berg-Kirkpatrick. 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 (Volume 1: Long Papers)*, pp. 1661–1671, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1154. URL https://aclanthology.org/P18-1154/.

Matt Gardner Johannes Welbl, Nelson F. Liu. Crowdsourcing multiple choice science questions. 2017.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. In *arXiv:1702.08734*, 2017. URL https://arxiv.org/abs/1702.08734.

Kagi Search. llm-chess-puzzles: Benchmark llm reasoning capability by solving chess puzzles. GitHub repository, April 2025. URL https://github.com/kagisearch/llm-chess-puzzles. 1000 FEN puzzles; updated 2025-04-26. Accessed 2025-09-16.

Daniel Khashabi, Snigdha Chaturvedi, Michael Roth, Shyam Upadhyay, and Dan Roth. Looking beyond the surface:a challenge set for reading comprehension over multiple sentences. In *NAACL*, 2018.

Tushar Khot, Peter Clark, Michal Guerquin, Peter Jansen, and Ashish Sabharwal. Qasc: A dataset for question answering via sentence composition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8082–8090, 2020.

Jaechang Kim, Jinmin Goh, Inseok Hwang, Jaewoong Cho, and Jungseul Ok. Bridging the gap between expert and language models: Concept-guided chess commentary generation and evaluation. *arXiv preprint arXiv:2410.20811*, 2024. URL https://arxiv.org/abs/2410.20811.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Andrew Lee, David Wu, Emily Dinan, and Mike Lewis. Improving chess commentaries by combining language models with symbolic reasoning engines. *arXiv preprint arXiv:2212.08195*, 2022. URL https://arxiv.org/abs/2212.08195.

Andrew Lee, Antonio Gulli, Bo Chang, Bob Fraser, Bovard Doerschuk-Tiberi, Chad Woodford, Chris Prichard, Chuck Sugnet, Daniel Hennes, Dima Yeroshenko, DJ Sterling, Elsa Dong, Hann Wang, Harrison Jobe, Ian Gemp, Jaimie Hwang, Jie Ren, John Schultz, Jon Lipovetz, Jun Peng, Justin Chiu, Karim Hakimzadeh, Kate Olszewska, Laurel Prince, Lloyd Hightower, Marc Lanctot, Martyna Plomecka, Meg Risdal, Meghan O'Connell, Minmin Chen, Nate Keating, Nenad Tomasev, Oran Kelly, Orhan Firat, Phoebe Kirk, Riley Jones, Roxanne Daniel, Ryan Trostle, Sahand Sharifzadeh, Siqi Liu, Timothy Chung, Tom Mason, Will Cukierski, Ya Xu, Yao Yan, Yi Su, Yuchen Zhuang, Yuting Han, and Yuexiang Zhai. Chess Text Input. https://www.kaggle.com/benchmarks/kaggle/chess-text, 2025. Google DeepMind, Google Cloud, Kaggle.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. *arXiv preprint arXiv:1906.00300*, 2019.

Leela Chess Zero Project. Lc0 releases. GitHub releases, 2025. URL https://github.com/LeelaChessZero/lc0/releases. Accessed: 2025-09-16.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1677–1687, Virtual Event, CA, USA, 2020. ACM. doi: 10.1145/3394486.3403219. URL https://dl.acm.org/doi/10.1145/3394486.3403219.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Sewon Min, Julian Michael, Hannaneh Hajishirzi, and Luke Zettlemoyer. Ambigqa: Answering ambiguous open-domain questions. *arXiv preprint arXiv:2004.10645*, 2020.

Yu Nasu. Efficiently updatable neural-network-based evaluation function for computer shogi. Technical report (English translation), 2018. URL https://oscarbalcells.com/assets/nnue_paper_english.pdf. Original in Japanese; English translation used. Accessed: 2025-09-16.

OpenAI. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024a.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence, 2024b. URL https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

OpenAI. Openai o1 system card, December 2024c. URL https://cdn.openai.com/o1-system-card-20241205.pdf.

OpenAI. Gpt-4.5 system card, 2025a. URL https://cdn.openai.com/gpt-4-5-system-card-2272025.pdf.

OpenAI. Gpt-5 system card, 2025b. URL https://cdn.openai.com/pdf/8124a3ce-ab78-4f06-96eb-49ea29ffb52f/gpt5-system-card-aug7.pdf.

OpenAI. Openai o3-mini system card, February 2025c. URL https://cdn.openai.com/o3-mini-system-card-feb10.pdf.

OpenAI. Openai o3 and o4-mini system card, 2025d. URL https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf.

OpenRouter. Openrouter: One api for any model. https://openrouter.ai, 2023.

Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. Quality: Question answering with long input texts, yes! *arXiv preprint arXiv:2112.08608*, 2021.

python chess. Pgn parsing and writing (documentation). https://python-chess.readthedocs.io/en/latest/pgn.html, 2024.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2124. URL https://aclanthology.org/P18-2124/.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.

Maxim Saplin. Llm chess leaderboard. Project website, 2025. URL https://maxim-saplin.github.io/llm_chess/. Elo vs. calibrated engine and random baselines. Accessed 2025-09-16.

Claude E Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. URL https://arxiv.org/abs/1712.01815.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/10.1126/science.aar6404.

Stockfish Developers. Stockfish 12. Stockfish Blog, 2020a. URL https://stockfishchess.org/blog/2020/stockfish-12/. Accessed: 2025-09-16.

Stockfish Developers. Introducing nnue evaluation. Stockfish Blog, 2020b. URL https://stockfishchess.org/blog/2020/introducing-nnue-evaluation/. Accessed: 2025-09-16.

Stockfish Developers. Stockfish 17.1. Stockfish Blog, 2025. URL https://stockfishchess.org/blog/2025/stockfish-17-1/. Accessed: 2025-09-16.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Zhenwei Tang, Difan Jiao, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. Maia-2: A unified model for human-ai alignment in chess. *Advances in Neural Information Processing Systems*, 37:20919–20944, 2024.

Zhenwei Tang, Difan Jiao, Eric Xue, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. Learning to imitate with less: Efficient individual behavior modeling in chess. *arXiv preprint arXiv:2507.21488*, 2025a.

Zhenwei Tang, Difan Jiao, Blair Yang, and Ashton Anderson. Seam: Semantically equivalent across modalities benchmark for vision-language models. *arXiv preprint arXiv:2508.18179*, 2025b.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Lichess Team. Lichess open database — evaluations. https://database.lichess.org/#evaluations, 2025a. 288,977,589 positions with Stockfish PVs and cp; schema documented on page. Last updated 2025-09-19.

Lichess Team. Lichess open database — puzzles. https://database.lichess.org/#puzzles, 2025b. 5,311,149 puzzles; rated and tagged; CC0. Last updated 2025-09-19.

Mistral AI team. Large enough. 2024. URL https://mistral.ai/news/mistral-large-2407.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. ♫ MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. doi: 10.1162/tacl_a_00475. URL https://aclanthology.org/2022.tacl-1.31/.

Alan M. Turing. Digital computers applied to games. In B. V. Bowden (ed.), *Faster than Thought: A Symposium on Digital Computing Machines*, pp. 286–310. Pitman, London, 1953. Chess section pp. 288–295.

Shu Wang, Lei Ji, Renxi Wang, Wenxiao Zhao, Haokun Liu, Yifan Hou, and Ying Nian Wu. Explore the reasoning capability of llms in the chess testbed. *arXiv preprint arXiv:2411.06655*, 2024a. URL https://arxiv.org/abs/2411.06655.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024b.

Qianfeng Wen, Yifan Liu, Joshua Zhang, George Saad, Anton Korikov, Yury Sambale, and Scott Sanner. Elaborative subtopic query reformulation for broad and indirect queries in travel destination recommendation. *arXiv preprint arXiv:2410.01598*, 2024.

Qianfeng Wen, Yifan Liu, Justin Cui, Joshua Zhang, Anton Korikov, George-Kirollos Saad, and Scott Sanner. A simple but effective elaborative query reformulation approach for natural language recommendation. *arXiv preprint arXiv:2510.02656*, 2025.

Wikipedia contributors. Forsyth–edwards notation. https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation, 2025. Accessed 2025-09-22.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.

Hongyu Zang, Zhiwei Yu, and Xiaojun Wan. Automated chess commentator powered by neural chess engine. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 5952–5961, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1597. URL https://aclanthology.org/P19-1597/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv:2506.05176*, 2025. URL https://arxiv.org/abs/2506.05176.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*, 2023.

A USE OF LARGE LANGUAGE MODELS STATEMENT

Large language models were utilized exclusively for improving the clarity of writing and correcting grammatical errors. They were not employed for generating research ideas or influencing the intellectual substance of this work.

B ADDITIONAL RESULTS

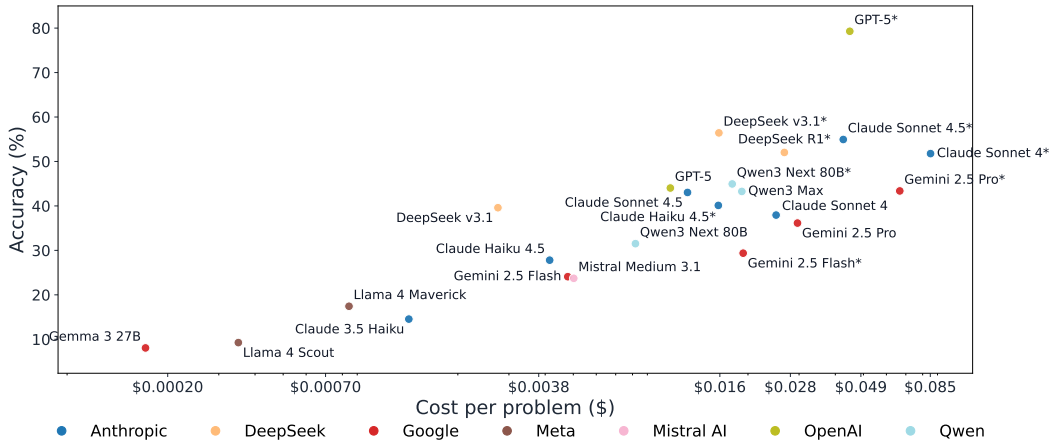
Table 4: Effect of prompt context on overall accuracy.

	Base	**ΔExample**	**+PieceArr**
Claude 3.5 Haiku	14.5%	14.5%	**23.5%**
Gemini 2.5 Flash	24.1%	23.4%	**31.0%**

Benchmark Robustness. We gave format examples in the prompts in order to extract final answers from LLMs' responses. As shown in Table 4, when a different set of format examples is given ($\Delta Example$), model performance is mostly consistent with the results when the default format examples are used (Base). Such results demonstrate the robustness of ChessQA with regard to variant format examples, and our curated prompts ensure that models do not use format examples as in-context learning examples.

Cost Effectiveness. We evaluated models in terms of accuracy per dollar spent in Fig 6, highlighting significant disparities driven by varying model architectures and pricing structures. The cost per problem ranged from as low as $0.0002 to as high as $0.09, a difference spanning over 500 times. Gemma 3 27B emerged as the most cost-effective model, achieving approximately 480 accuracy points per dollar, despite its relatively low absolute accuracy. Conversely, top-performing models like GPT-5* demonstrated superior accuracy but at a significantly higher cost, emphasizing a trade-off between raw performance and economic efficiency. It is important to note that these cost evaluations are subject to slight inaccuracies due to routing variability in model queries.

Figure 6: Performance comparison w.r.t cost per problem. * denotes thinking enabled.



Token Efficiency. In Fig 7, we provide detailed insights into token consumption per task across different categories. Reasoning-enabled models consistently utilize significantly more tokens, with GPT-5* notably using up to 14,823 tokens per task in Position Judgment and 11,668 tokens in Short Tactics. Base models generally consume fewer tokens, reflecting a reduced depth in their reasoning process. Token usage varies substantially by category, with Position Judgment and Short Tactics consistently demanding higher token counts due to their complexity and the need for intricate reasoning paths. Conversely, Structural and Semantic tasks show more modest token usage, reflecting their relative ease and straightforward nature.

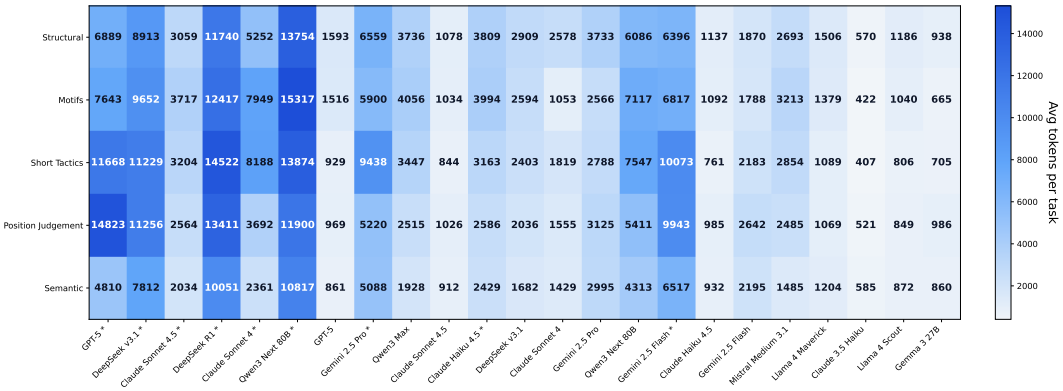


Figure 7: The overall and per-category tokens used comparison. * denotes thinking enabled.

Model Agreement. Fig 8 presents cross-model agreement across all evaluated tasks, measured by average accuracy overlap. High-performing models like GPT-5*, DeepSeek v3.1*, and Claude Sonnet 4* exhibit strong mutual agreement (up to 74.6%), indicating similar reasoning strategies and correct solution pathways. Conversely, lower-performing models demonstrate significantly less agreement, often below 50%. The heatmap clearly reveals clusters of higher agreement among top-tier models, suggesting that advanced reasoning capabilities converge on similar solutions, whereas divergence among lower-tier models highlights variability and inconsistency in basic tactical comprehension.

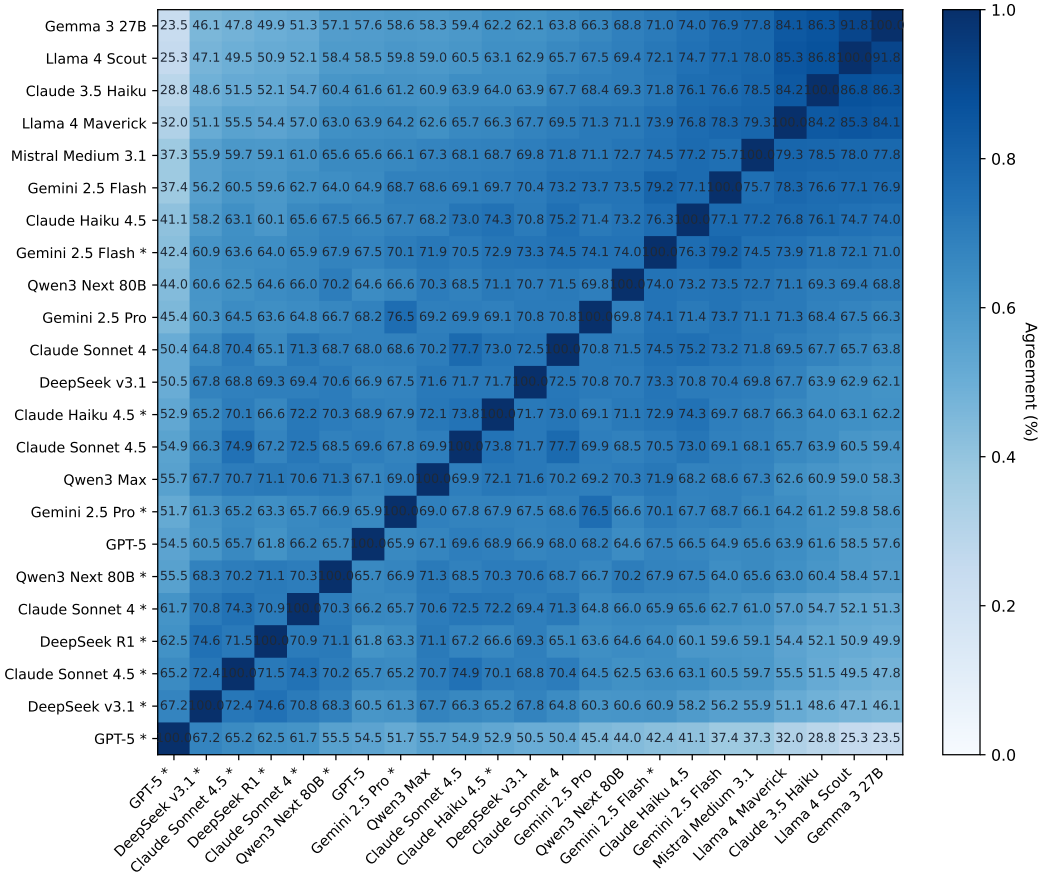


Figure 8: Cross-model agreement on all tasks, shown by average accuracy.

C TASK DESIGN DETAILS

C.1 *Structural*

Scope and answer format. *Structural* targets rule-level competence only: legal move generation, check relations, attack/control/protect sets, full piece listing, and deterministic state updates from UCI moves. Prompts allow free-form reasoning but require a single terminal line:

`FINAL ANSWER: <answer>`

We provide type-specific format exemplars inside each prompt to reduce formatting errors; prompts are constructed with fixed format examples at build time. Gold answers are canonicalized at generation (sorted lists, fixed piece-order), enabling exact-match scoring on the terminal line.

Subtask specifications.

- **Piece arrangement**: Given a FEN, list all pieces in a fixed order (White: K,Q,R,B,N,P; then Black: K,Q,R,B,N,P). For each piece type, squares are alphabetized.
- **Legal moves (piece/all)**: Enumerate legal UCI moves from the position; the per-piece variant conditions on a target square.
- **Check detection**: If the side to move is in check, list all checking pieces as *"Color Piece at <square>"*.
- **Check-in-1**: List all legal moves that give immediate check.
- **Capture / Control / Protect squares**: For a specified *non-pawn, non-king* piece and square, return, respectively: *enemy-occupied* attacked squares; *empty* attacked squares; and *friendly-occupied* attacked squares *excluding the king*. If the piece is pinned, the set is empty.
- **State tracking (short/mid/long)**: Apply a sequence of UCI moves (lengths 1–5 / 6–10 / 11–15) to a start FEN and output the exact resulting FEN.

Generation pipeline. We build eight puzzle-based subtasks from a shuffled Lichess puzzle CSV, taking at most one subtask per puzzle ID to prevent reuse; state-tracking items are extracted from broadcast PGNs after the first 30 half-moves. Task builders and prompt constructors follow the structural generation script and utilities. Full algorithm showed by Algo. 1.

Canonicalization and scoring.

- **Evaluation:** exact string match on the terminal line after extracting the substring following `FINAL ANSWER:`.
- **Sorted outputs:** multi-item answers are sorted at gold construction (e.g., legal moves, check-in-1, capture/control/protect).
- **Check detection:** checking pieces are returned in the chess library's attacker enumeration order (no additional sort).
- **Arrangement:** piece group order is fixed; per-type square lists are alphabetized.

Prompts and exemplars. We show prompts of the tasks receptively.

Piece arrangement

```
You are given a chess position in FEN: <FEN>.
Provide the complete piece arrangement of this position. List all
↪  pieces with their colors, types, and squares.
Format: 'Color PieceType: [square1, square2, ...]', separated by
↪  commas and spaces for different piece types.
List the pieces in the order of White pieces first (King, Queen,
↪  Rook, Bishop, Knight, Pawn) followed by Black pieces (King,
↪  Queen, Rook, Bishop, Knight, Pawn).
```

Algorithm 1 Structural Tasks Generation

Inputs: Lichess puzzle CSV (FEN,PuzzleId), broadcast PGN file
Params: $N_{\text{sample}} = 100$ per subtask; seed $= 42$
SEEDEVERYTHING(42); $D \leftarrow$ READPUZZLES(); shuffle once
$U \leftarrow \emptyset$ ▷ used puzzle IDs
Initialize counters for the 11 subtasks to 0
for row in D **do**
  if row.PuzzleId $\in U$ **then continue**
  end if
  $B \leftarrow$ BOARD(row.FEN)
  for `task` $\in$ {arrangement, legal_piece, legal_all, check_det, check_in_1, capture, protect, control} **do**
    if counter[`task`] $\geq N_{\text{sample}}$ **then continue**
    end if
    $t \leftarrow$ GENERATE_`task`$(B, \dots)$ ▷ returns NONE if preconditions fail
    if $t \neq$ NONE **then**
      append t; counter[`task`]++; $U \leftarrow U \cup$ {row.PuzzleId}; **break**
    end if
  end for
  if all eight puzzle-based counters $\geq N_{\text{sample}}$ **then break**
  end if
end for
// State tracking from PGNs
for game in READPGN() **do**
  if all {short, mid, long} counters $\geq N_{\text{sample}}$ **then break**
  end if
  for (subtype, min, max) in {(short,1,5),(mid,6,10),(long,11,15)} **do**
    if counter[subtype] $\geq N_{\text{sample}}$ **then continue**
    end if
    $L \leftarrow$ random integer in $[min, max]$
    (start_fen, moves, game_id) $\leftarrow$ EXTRACTFRAGMENT(game, start_after=30, track=L)
    if moves **then**
      $t \leftarrow$ MAKESTATETRACKING(game_id, start_fen, moves, subtype)
      append t; counter[subtype]++; **break**
    end if
  end for
end for
SAVETASKS(`structural.jsonl`)

```
If a piece type has no pieces on the board, skip it in the
↪  listing.
List the squares for each piece type in alphabetical order.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: White King: ['e1'], White Queen: ['d1'],
↪  White Rook: ['a1', 'h1'], White Bishop: ['c1', 'f1'], White
↪  Knight: ['b1', 'g1'], White Pawn: ['a2', 'b2', 'c2', 'd2',
↪  'e2', 'f2', 'g2', 'h2'], Black King: ['e8'], Black Queen:
↪  ['d8'], Black Rook: ['a8', 'h8'], Black Bishop: ['c8', 'f8'],
↪  Black Knight: ['b8', 'g8'], Black Pawn: ['a7', 'b7', 'c7',
↪  'd7', 'e7', 'f7', 'g7', 'h7']
```

Legal moves (piece)

```
You are given a chess position in FEN: <FEN>.
Find all legal moves for the piece on square <square>. List the
↪  moves in UCI format, separated by commas and spaces.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e2e3, e2e4
```

Legal moves (all)

```
You are given a chess position in FEN: <FEN>.
Find all legal moves in this position. List the moves in UCI
↪  format, separated by commas and spaces.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e2e4, c2b1q
```

Check detection

```
You are given a chess position in FEN: <FEN>.
In this position, the side to move is in check. Identify the
↪  piece(s) that is delivering the check.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
List each checking piece with its color, type, and square (e.g.,
↪  White Queen at e5). Separate multiple pieces with commas and
↪  spaces if applicable.
```

Check-in-1

```
You are given a chess position in FEN: <FEN>.
Find all moves that put the opponent in check. List the moves in
↪  UCI format, separated by commas and spaces.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e2e4, c2b1q
```

Capture squares

```
You are given a chess position in FEN: <FEN>.
Find all squares that the <Color Piece> on <square> can capture
↪  (i.e. every square that has an opponent piece such that the
↪  <Color Piece> on <square> could legally move to that square
↪  and capture the piece).
Exclude captures if the <Color Piece> on <square> is pinned to its
↪  king and thus cannot move.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e4, f5
```

Control squares

```
You are given a chess position in FEN: <FEN>.
Find all squares that the <Color Piece> on <square> controls (i.e.
↪  every empty square that the <Color Piece> on <square> could
↪  legally move to, excluding squares occupied by any piece).
Exclude control if the <Color Piece> on <square> is pinned to its
↪  king and thus cannot move.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e4, f5
```

Protect squares

```
You are given a chess position in FEN: <FEN>.
```

```
Find all squares that contain pieces that the <Color Piece> on
↪  <square> protects (i.e. every square that contains a piece
↪  such that the <Color Piece> on <square> could legally
↪  recapture if an enemy piece captured it, excluding the king
↪  since it can't be captured).
Exclude protection if the <Color Piece> on <square> is pinned to
↪  its king and thus cannot move.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: e4, f5
```

State tracking—short

```
Given an initial FEN and a sequence of UCI moves, apply the moves
↪  in order and output the exact resulting FEN.
Initial FEN: <start FEN>
Moves (UCI): <u1 u2 ...>
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
↪  w KQkq - 0 1
```

State tracking—mid

```
Given an initial FEN and a sequence of UCI moves, apply the moves
↪  in order and output the exact resulting FEN.
Initial FEN: <start FEN>
Moves (UCI): <u1 u2 ...>
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
↪  w KQkq - 0 1
```

State tracking—long

```
Given an initial FEN and a sequence of UCI moves, apply the moves
↪  in order and output the exact resulting FEN.
Initial FEN: <start FEN>
Moves (UCI): <u1 u2 ...>
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Example final answer: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR
↪  w KQkq - 0 1
```

C.2 *Motifs*

Scope and answer format. *Motifs* targets pattern-level tactics that do not require deep search: absolute pins, skewers, forks, batteries, and enumerating discovered-check and double-check moves. Prompts permit free-form reasoning but require a single terminal line:

```
FINAL ANSWER: <answer>
```

Gold labels come from deterministic detectors (ray scans and legal-move simulation) and are serialized as canonical strings; multi-item answers are comma-separated.

Subtask specifications.

- **Pins (absolute)** — Return every triplet `pinning>pinned>target` using *squares*; only absolute pins (target is the king) are included.
- **Skewers** — Every `attacker>front>back` where the front piece is *more valuable* than the back piece (values: P=1, N=B=3, R=5, Q=9, K=100).
- **Forks** — For each forking square, list attacked enemy pieces as `forking>sq1-sq2(-sq3...)`. Inside each fork, target squares are sorted `a`→`h`, then `1`→`8`.
- **Batteries** — For each aligned same-color slider group on one line with empty squares between, return `sq>sq(>sq...)`; multiple batteries comma-separated. (Emission follows a deterministic board-scan order.)
- **Discovered checks** — List all UCI moves by the side to move that uncover a rook/bishop/queen check on the enemy king.
- **Double checks** — List all UCI moves that produce two simultaneous checkers after the move.

(Detectors for each subtask are implemented on top of `python-chess` with explicit line/ray scans and legal-move simulation (Fiekas, 2024b).)

Generation pipeline. We stream shuffled Lichess puzzles (FEN,PuzzleId), emit at most one motif item per puzzle ID, and continue until each of the six subtasks reaches its target count (default 100). For each board we run the corresponding detector(s); when a detector yields at least one motif (or checking move), we serialize the answer in the task's canonical format and stop for that puzzle ID. Full algorithm showed by Algo. 2.

Algorithm 2 *Motifs* Tasks Generation

Inputs: Lichess puzzle CSV (FEN, PuzzleId) **Params:** N_{sample}=100, seed= 42
SEEDEVERYTHING(42); $D \leftarrow$ READPUZZLES()
$U \leftarrow \emptyset$ ▷ used puzzle IDs
counts[`pin`, `fork`, `battery`, `skewer`, `discovered_check`, `double_check`] $\leftarrow 0$
for row $\in D$ **do**
  if row.PuzzleId $\in U$ **then continue**
  end if
  $B \leftarrow$ BOARD(row.FEN)
  for g $\in$ {PIN, FORK, BATTERY, SKEWER, DISCOVEREDCHECK, DOUBLECHECK} **do**
    if counts[g] $\geq N_{\text{sample}}$ **then continue**
    end if
    $t \leftarrow$ GENERATE_$g(B, \dots)$ ▷ returns NONE if no motif
    if $t \neq$ NONE **then**
      append t; counts[g]++; $U \leftarrow U \cup$ {row.PuzzleId}; **break**
    end if
  end for
  if all counts $\geq N_{\text{sample}}$ **then break**
  end if
end for
SAVETASKS(`motif.jsonl`)

Detector definitions. **Pins.** From every enemy rook/bishop/queen, scan rays; if the first two pieces on a ray are ours and the second is our king, record `pinning>pinned>target`.

Skewers. For each friendly slider, trace an attack ray: if the first enemy piece (*front*) is strictly more valuable than a second enemy piece (*back*) behind it on the same line with no interveners, record `attacker>front>back`.

Forks. Any piece that simultaneously attacks $\geq$ 2 enemy pieces yields one fork: `forking>sq1-sq2(-sq3...)` with target squares sorted.

Batteries. Enumerate ranks, files, and diagonals; whenever $\geq$ 2 same-color sliders appear on a line with empty squares between them (and line-consistent movement), emit the ordered group; deduplicate groups.

Discovered checks. Simulate each legal move; if the resulting position is check and some checker is a slider *other than* the moved piece whose line to the enemy king was previously blocked only by the moved piece, record the UCI move.

Double checks. Simulate each legal move; if the resulting position has at least two distinct checkers, record the UCI move.

Canonicalization and scoring.

- **Answer line:** exact match on `FINAL ANSWER: <answer>`.
- **Separators:** use > within a structure (pin/skewer/battery chain), − within a fork's target list, and comma+space between multiple motifs/moves.
- **Ordering:** fork targets explicitly sorted (file a→h, then rank 1→8); discovered/double checks output as UCI; other groups follow the detector's deterministic board-scan order. (Typical LLM errors we observed include extra/omitted items and format slips.)

Prompts and exemplars. We show prompts of the tasks receptively.

Pins

```
You are given a chess position in FEN: <FEN>.
Identify all absolute pins in this position. An absolute pin
↪  occurs when a piece cannot move because it would expose its
↪  own king to check.
For each pin, provide the key squares in the format:
↪  pinning_piece>pinned_piece>target_piece (e.g., d1>d7>d8).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: d1>d7>d8, a2>e2>h2
```

Skewers

```
You are given a chess position in FEN: <FEN>.
Identify all skewers in this position. A skewer occurs when a more
↪  valuable piece is attacked first and forced to move, exposing
↪  a less valuable piece behind it to be captured.
For each skewer, provide the key squares in the format:
↪  skewering_piece>front_piece>back_piece (e.g., a5>e5>h5).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: a5>e5>h5, h1>h4>h7
```

Forks

```
You are given a chess position in FEN: <FEN>.
Identify all forks in this position. A fork occurs when one piece
↪  attacks two or more enemy pieces simultaneously.
For each fork, provide the key squares in the format:
↪  forking_piece>attacked_piece1-attacked_piece2(-attacked_piece3
↪  ...) (e.g., e5>e7-f6).
Order attacked pieces alphabetically (a>h, then 1>8).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: e5>e7-f6, f3>e1-g1
```

Batteries

```
You are given a chess position in FEN: <FEN>.
Identify every battery (2 or more aligned long-range pieces, e.g.,
↪  RR/RQ/QQ on files or ranks; BB/BQ/QQ on diagonals; same color;
↪  no pieces between).
Report each battery as the squares of the pieces in alphabetical
↪  order (a>h, 1>8), using '>' to separate squares in a battery
↪  and ',' to separate multiple batteries (e.g., h1>h4).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: b2>e5>h8, h1>h7
```

Discovered checks

```
You are given a chess position in FEN: <FEN>.
Identify all discovered-check moves (your move uncovers a check
↪  from a rook/bishop/queen on the enemy king).
Report each as UCI move (e.g., e2e4).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: e2e4, c2b1q
```

Double checks

```
You are given a chess position in FEN: <FEN>.
Identify all moves that deliver a double check (two pieces give
↪  check after the move).
Report each as UCI move (e.g., g1f3).
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
If more than one, separate with a comma and a space.
Example final answer: g1f3, a2a1q
```

C.3 *Short Tactics*

Scope and answer format. *Short Tactics* consists of short, single–move tactics drawn from Lichess puzzles. Each item asks for the *best move* for the side to play, output as a single UCI move on the terminal line:

```
FINAL ANSWER: <answer>
```

All prompts are constructed and ship with UCI exemplars. Gold answers are taken directly from the puzzle principal variation (PV) after a preparatory "pre-move" step (as noted by Lichess puzzle database on how to preprocess data, see `https://database.lichess.org/#puzzles`).

Subtask specifications. We provide two families of subtasks:

- **Rating-split**: four levels determined by the puzzle rating — `beginner` ($\leq$ 999), `intermediate` ($\leq$ 1499), `advanced` ($\leq$ 1999), and `expert` ($\geq$ 2000).
- **Theme-split**: per-theme best-move tasks. Themes including: fork, exposedKing, attraction, discoveredAttack, sacrifice, defensiveMove, intermezzo, pin, mateIn1, smotheredMate, zugzwang, mateIn2, capturingDefender, backRankMate, xRayAttack, skewer, hangingPiece, mateIn3, advancedPawn, queensideAttack, trappedPiece, promotion, deflection and doubleCheck, in total 20 themes.

Generation pipeline. A puzzle row is eligible only if it passed rate limit filter, popularity filter and its PV length is under our threshold. Detailed algorithm is showed by Algo 3.

Algorithm 3 *Short Tactics* generation

Inputs: puzzle CSV; theme list JSON **Params:** thresholds for filters
SEEDEVERYTHING(seed); $D \leftarrow$ READPUZZLES()
$U \leftarrow \emptyset$ *(used PuzzleIds)*
// Rating split
for row in D (shuffled) **do**
  if FAILFILTERS(row) or row.PuzzleId $\in U$ **then continue**
  end if
  $(fen, move) \leftarrow$ MAKEPREMOVE(row)
  $\ell \leftarrow$ RATINGTOLEVEL$(row$.Rating$)$
  if count$[\ell] < N_{\text{sample_rating}}$ **then**
    emit TASK(`tactic_rating_`ℓ, fen, move, row.*); $U \leftarrow U \cup \{row.PuzzleId\}$
  end if
  if all levels filled **then break**
  end if
end for
// Theme split
$T \leftarrow$ set of allowed themes (JSON)
for row in D (continue) **do**
  if FAILFILTERS(row) or row.PuzzleId $\in U$ **then continue**
  end if
  $\mathcal{I} \leftarrow$ themes(row) $\cap T$;
  if $\mathcal{I} = \emptyset$ **then continue**
  end if
  for $t \in \mathcal{I}$ **do**
    if count$[t] < N_{\text{sample_theme}}$ **then**
      $(fen, move) \leftarrow$ MAKEPREMOVE(row)
      emit TASK(`tactic_theme_`t, fen, move, row.*, primary_theme=t); $U \leftarrow U \cup \{row.PuzzleId\}$
      break
    end if
  end for
  if all themes filled **then break**
  end if
end for
SAVETASKS(`tactic.jsonl`)

Canonicalization and scoring.

- **Answer type**: `single` — one UCI move exactly (include promotion suffix, e.g., `d7d8q`, if applicable).
- **Evaluation**: exact string match on the terminal line after `FINAL ANSWER:`.
- **Formatting**: no extra tokens, punctuation, or SAN; UCI only. Prompts include exemplars to reduce formatting errors.

Prompts and exemplars. We show prompts of two families respectively.

Best move (rating-split)

```
You are given a chess position in FEN: <FEN>.
Find the best move for the side to play.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Use UCI notation (e.g., e2e4, c2b1q) for the final answer.
```

Best move (theme-split)

```
You are given a chess position in FEN: <FEN>.
Find the best move for the side to play.
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Use UCI notation (e.g., e2e4, c2b1q) for the final answer.
```

C.4 *Position Judgment*

Scope and answer format. Class 4 asks models to judge a static position's strength in *centipawns from White's perspective*. For each item we present a FEN and a fixed 5-choice option set; the model must output exactly one number (as text) on the terminal line `FINAL ANSWER: <answer>`. Gold labels are derived from engine evaluations and mapped to the nearest option. This category's items are all single answer questions.

Categories and option set. We bucket positions by their centipawn (cp) score into five disjoint ranges (White's perspective), and we always show the same choice list $\{-400, -200, 0, 200, 400\}$.

Category	cp range	Task type
`losing`	$[-450, -350]$	`losing`
`disadvantage`	$[-250, -150]$	`disadvantage`
`neutral`	$[-50, 50]$	`neutral`
`advantage`	$[150, 250]$	`advantage`
`winning`	$[350, 450]$	`winning`

At label time we snap the engine's cp to the *closest* option; ties break toward the first option in the list (e.g., $+100 \to 0$, $-100 \to -200$).

Gold construction (engine stream → tasks). We process the evaluation stream from Lichess eval database (`https://database.lichess.org/#evals`) and parse each line to extract: the FEN, the *deepest* entry in `evals` (by `depth`), and its first principal variation `pvs[0]` (skipping lines that indicate mate). From that PV we take the centipawn score (`cp`), the PV line (`line`), and metadata such as `depth` and `knodes`. We then compute a position hash from the first four FEN fields (piece placement, side to move, castling, en passant) to deduplicate and bucket the position into one of the five categories above. For each category we sample up to `tasks_per_category` positions and create one task per position.

Canonicalization & scoring. Prompts allow free-form reasoning, but the grader expects a *single* numeral chosen from the option set on the terminal line. The correct answer is the nearest option to the engine cp; we attach the exact engine cp and PV depth in the item metadata.

Prompt.

```
You are analyzing a chess position in FEN: <FEN>.
Estimate the Stockfish evaluation in centipawns (from White's
↪  perspective). Think deeper about this position: Don't just
↪  evaluate the current board state. Consider what the most
↪  likely moves are for both sides and how the centipawn
↪  evaluation would change as the position develops. Analyze a
↪  moves ahead - what does the future of this position look like?
↪  How would a strong engine assess this position after
↪  calculating many moves deep?
Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <answer>
Choose the closest evaluation from the following options: -400,
↪  -200, 0, 200, 400.
Example final answers: 400
```

Algorithm. We have two steps, first select position, then create the problem. Detailed algorithm is showed below.

Algorithm 4 *Position Judgment* — Position Selection

Input: stream of engine evaluations on positions; target count T per category
Categories: five disjoint cp ranges (White's view): `losing`, `disadvantage`, `neutral`, `advantage`, `winning` (see main text)
Output: balanced set $\mathcal{S}$ of positions with category labels
Initialize empty buckets $\mathcal{B}[c]$ for each category c; initialize empty set of seen position IDs
for each evaluation record **do**
    Extract the deepest available engine evaluation (centipawns cp and principal line); skip mate scores
    Normalize the position ID from the FEN's piece/turn/castling/en-passant fields; **continue** if already seen
    $c \leftarrow \textsc{CategoryFromCP}(cp)$; **continue** if c is undefined
    if $|\mathcal{B}[c]| < T$ **then**
        append record to $\mathcal{B}[c]$; mark position ID as seen
    end if
    if all categories have $|\mathcal{B}[c]| = T$ **then break**
    end if
end for
return $\mathcal{S} \leftarrow \bigcup_c \mathcal{B}[c]$

Algorithm 5 *Position Judgment* — Item Assembly

Input: balanced set $\mathcal{S}$ of labeled positions
Constant option set: $\mathcal{O} = \{-400, -200, 0, 200, 400\}$
Output: multiple-choice judgement items with exact-match answers
for each position $x \in \mathcal{S}$ with engine $cp(x)$ and category $c(x)$ **do**
    $a(x) \leftarrow \arg\min_{o \in \mathcal{O}} |cp(x) - o|$ ▷ nearest option snap; ties break by fixed option order
    Build the prompt with the shared prompt constructor, appending the option list $\mathcal{O}$ and the terminal line format `FINAL ANSWER: <answer>`
    Store one item with fields: *type*=`judgement_`$c(x)$, *input*=FEN, *options*=$\mathcal{O}$, *answer*=$a(x)$, and metadata (cp, depth, principal line)
end for

C.5 *Semantic*

Scope and task family. *Semantic* evaluates *semantic understanding* by asking a model to pick, from several alternatives, the natural-language commentary that best describes a concrete chess *position and the just-played move*. Each item shows: (i) a FEN snapshot *before* the move, (ii) the move in UCI, and (iii) 4 textual options (one gold, three distractors). The model must output a single letter `A–D` on the terminal line `FINAL ANSWER: <letter>`. Items are constructed from real human comments extracted from PGN mainlines and then curated via a cleaning and relevance-judgment pipeline.

Pipeline overview (data → tasks).

1. **Extract & keyword-gate comments** from PGN mainlines. For each commented move we keep: raw comment, `move_uci`, FENs before/after, SAN history up to the move, move number, side to move, moved piece, players, and lightweight *thematic keywords* (e.g., *fork*, *outpost*, *zugzwang*). We only keep comments with ≥ 5 words that match at least one chess keyword (default keyword list provided).
2. **Clean comments** with an offline LLM (`Qwen/Qwen3-30B-A3B-Instruct-2507`, inferenced by vllm (Yang et al., 2025; Kwon et al., 2023)) pass through: replace mentions of the two players with "White/Black"; if any *other* person name appears, `SKIP`; normalize figurines/markup; optionally convert first-person pronouns when the annotator equals a player. Output is either the cleaned text or `SKIP`.
3. **Judge relevance** with a second LLM (`Qwen/Qwen3-30B-A3B-Instruct-2507`, inferenced by vllm (Yang et al., 2025; Kwon et al., 2023)) pass (`KEEP`/`DROP`), preceded by a

chess-aware heuristic that quickly drops generic or non-chess content (regex for SAN/UCI, chess terms, square names). Only `KEEP` items continue.

4. **Assemble MCQ tasks** for four *distractor strategies*: `easy_random`, `keyword`, `piece_stage`, `embedding`. We compute dense embeddings once (cached) to support semantic neighbors; options are shuffled and mapped to a letter label with exact-match grading.

Prompt.

```
You are given a chess position in FEN: <FEN_before>
A player makes the move: <move_uci>

Select the commentary that best describes this position and move.

Options:
A. <option A>
B. <option B>
C. <option C>
D. <option D>

Analyze step by step and explain your reasoning.
Finish with a single line formatted EXACTLY as:
FINAL ANSWER: <letter>
```

Distractor strategies.

- `easy_random`: sample distractors uniformly from other cleaned+kept comments. Backfilled if collisions reduce uniqueness.
- `keyword`: pick comments sharing at least one thematic keyword with the gold; random fill if needed.
- `piece_stage`: match by (moved piece type, game phase bucket = opening/middlegame/endgame from move number) to increase surface plausibility; then fall back to same-piece or random.
- `embedding`: nearest neighbors by cosine similarity over sentence embeddings.

All strategies enforce *unique* options and ensure the gold string appears exactly once.

Configuration. We use `N_sample_mcq = 100` per variant, `num_options = 4`, `num_distractors = 3`, random sampling enabled, phase thresholds `opening=12`, `middlegame=30`, and embeddings via `Qwen/Qwen3-Embedding-8B` (Zhang et al., 2025) with normalized cosine retrieval (batched, cached).

Algorithm. Stage A — Comment extraction & keyword gating. (PGN $\rightarrow$ candidates) (Constructs per-move records, identifies moved piece, and formats PGN-until-move; keywords include tactics/strategy/endgame terms.)

Algorithm 6 ExtractCandidates

Input: PGN mainline games; chess keyword list; min word count m (default m=5)
Output: candidate set $\mathcal{C}$ of {comment, FEN_before/after, move_uci, SAN_so_far, move_no, side, moved_piece, players, keywords}
for each game G **do**
  for each mainline move node u with attached text t **do**
    compute KEYWORDS(t) using normalized matching; **continue** if none
    keep if WORDCOUNT$(t) \geq m$; record metadata from board state
  end for
end for
return $\mathcal{C}$

Stage C — Relevance judgment. (drop generic/irrelevant) (Combines fast heuristics and strict

Algorithm 7 JudgeRelevance

Input: $\mathcal{C}'$
$\mathcal{R} \leftarrow \emptyset$
Heuristic prefilter HEURISTICISRELEVANT(r): require chess tokens (SAN/UCI/squares or chess words); drop very short/generic text
LLM judge with context (FENs, move, PGN prefix); strict `KEEP`/`DROP`
for $r \in \mathcal{C}'$ **do**
    if not HEURISTICISRELEVANT(r) **then continue**
    end if
    $\ell \leftarrow$ LLMJUDGE(r)
    if $\ell =$ `KEEP` **then**
        add r to $\mathcal{R}$
    end if
end for
return $\mathcal{R}$

LLM labeling to retain comments that tie *directly* to the concrete move/position.)

Stage D — MCQ assembly with distractors.

Algorithm 8 BuildMCQ

Input: relevant comments $\mathcal{R}$; number of options n=4; distractors d=3; variants $\mathcal{V}$
$\mathcal{T} \leftarrow \emptyset$
Compute indices by keyword/piece/(piece, stage); compute or load embedding matrix (unit-normalized) for semantic neighbors
for each base item $i \in \mathcal{R}$ and variant $v \in \mathcal{V}$ **do**
    select d distractors via $v \in \{\text{random}, \text{keyword}, \text{piece\_stage}, \text{embedding}\}$ with fallbacks
    ensure uniqueness and pad from the pool if needed; shuffle options; map gold $\rightarrow \{A, \ldots, D\}$
    emit one task with prompt (`FEN_before`, `move_uci`, option list), letter key, and metadata; add to $\mathcal{T}$
end for
return $\mathcal{T}$

D EXAMPLE MODEL RESPONSES

We choose the best performed model GPT-5 * to display some example failures that models would have when doing problems.

D.1 WRONG PIECE / SQUARE RECOGNITION

Category / Task type: *Motifs* — Task type: skewer
Error type: Wrong Answer
Expected: `f7>g6>h5`
Model: `f7>f6>f4`

Question.

> You are given a chess position in FEN: `6k15qp1p2bp1Np1p1b3P5P2P3p1P11P5K2Q5 b - - 1 42`. Identify all skewers in this position. A skewer occurs when a more valuable piece is attacked first and forced to move, exposing a less valuable piece behind it to be captured. For each skewer, provide the key squares in the format: `skewering_piece>front_piece>back_piece` (e.g., FORMAT_EXAMPLE_PLACEHOLDER). Analyze step by step and explain your reasoning. Finish with a single line formatted EXACTLY as: FINAL ANSWER: `<answer>` If more than one, separate with a comma and a space.

Model response.

Step-by-step reasoning: - Identify line pieces: Black has a queen (f7) and bishops (d6,d5). White has a queen (c1). No rooks or bishops for White. - A skewer requires a line piece to attack a more valuable enemy piece that has a less valuable enemy piece directly behind it on the same line (with only empty squares in between). - Check Black queen f7: - Along the f-file downward: f6 (White knight) is attacked; behind it on the same file is f4 (White pawn). Knight (3) > Pawn (1), and the line is clear (f5 empty). This is a valid skewer: `f7>f6>f4`. - Other directions from f7 are blocked or do not produce a more-then-less value alignment. - Check Black bishops d6, d5: Their attacks hit either single pieces or alignments with equal/greater value behind (e.g., d6→f4→g3 are pawns; not a skewer), or are blocked. - Check White queen c1: Its rays are blocked (e3 pawn or b2 pawn) or hit no suitable pair of Black pieces. No skewer from White. Only one skewer exists. FINAL ANSWER: `f7>f6>f4`

D.2 *Short Tactics* LEGALITY MISTAKES

Category / Task type: *Short Tactics* — Task type: expert
Error type: Wrong Answer
Expected: `d7f6`
Model: `g7g5`

Question.

You are given a chess position in FEN: `3r1q1k/p2nNpp1/1p2p2p/8/nPP4Q/P4N2/5PPP/3R2K1 b - - 3 24`. Find the best move for the side to play. Analyze step by step and explain your reasoning. Finish with a single line formatted EXACTLY as: FINAL ANSWER: `<answer>` Use UCI notation (e.g., e2e4, c2b1q) for the final answer.

Model response.

Step-by-step reasoning:- Reconstructing the position: Black pieces include Kg8-h8, Qf8, Rd8, knights at a4 and d7, pawns a7, b6, e6, f7, g7, h6. White has Kg1, Qh4, Rd1, knights e7 and f3, pawns a3, b4, c4, f2, g2, h2.- Critical tactical motif: If Black moves the d7-knight (e.g., ...Ne5 or ...Nf6), White can reply Rxd8 Qxd8 and then Ng6+ followed by Qxd8, winning Black's queen. So moving the d7-knight is dangerous.- The e7-knight is guarded only by the queen on h4 along the diagonal h4–g5–f6–e7. If Black attacks the queen with ...g5, White must either: - Move the queen off that diagonal (e.g., Qg3/Qh5/Qf4), when Black plays ...Qxe7 winning the knight; or - Capture en passant-like with 25 Nxg5, but then ...Qxe7 again wins the e7-knight because the diagonal is blocked by the knight on g5. White may grab a pawn or two with Qxh6+ afterward, but Black remains up material.- Alternative attractive idea 24...Nc3 threatens Nxd1, and 25 Rc1? allows ...Ne2+ and ...Nxc1 winning the exchange, but White can avoid this with 25 Re1/Rf1, keeping things more balanced. Therefore 24...g5 is more forcing and clearly wins material.- Bad move refutation example: 24...Ne5? loses to 25 Rxd8 Qxd8 26 Ng6+ followed by Qxd8, winning Black's queen. Best move:- 24...g5! attacks the queen, disconnects it from supporting e7, and wins the e7-knight next move in all lines. FINAL ANSWER: g7g5

D.3 SOUND ANALYSIS, WRONG FINAL ANSWER

Category / Task type: *Short Tactics* — Task type: theme queensideAttack
Error type: Wrong Answer
Expected: `d7f6`
Model: `g7g5`

Question.

You are given a chess position in FEN: `6k14Rpp1p6pq3p31rp1P2PpPQ2PP1P71K5R b - - 0 30`. Find the best move for the side to play. Analyze step by step and explain your reasoning. Finish with a single line formatted EXACTLY as: FINAL ANSWER: `<answer>` Use UCI notation (e.g., e2e4, c2b1q) for the final answer.

Model response.

Let's decode the position. - White king: b1; queen: c3; rooks: e7 and h1; pawns: a2, b3, e4, f3, g3, h4. - Black king: g8; queen: a5; rooks: b4 and c4; pawns: a6, a3, d4, e5, f7, g7, h6. - Black to move. Key tactical feature: White's queen on c3 sits directly in front of Black's rook on c4. The move Rxc3 simply wins White's queen outright. There are no interferences or pins: the path from c4 to c3 is clear, and b4 continues to block the a5–c3 diagonal, so no self-pins or discovered checks arise. After 30...Rxc3, White cannot recapture the rook with the queen (it's gone), and any compensation ideas (like Re8+ or Rxf7) are completely insufficient; Black remains up a full queen with a safe king. Alternative tempting idea: 30...Rxb3+ 31.axb3 Qxc3 also wins the queen, but at the cost of sacrificing a rook first, leaving Black up a queen for a rook (smaller material gain) and giving White extra activity. Therefore, directly 30...Rxc3 is superior. Conclusion: The cleanest and strongest move is capturing the queen immediately. FINAL ANSWER: c4c3

D.4 SAYING "NO ANSWER" WHEN AN ANSWER EXISTS

Category / Task type: *Motifs* — Task type: skewer
Error type: Wrong Answer
Expected: `d7f6`
Model: `None`

Question.

You are given a chess position in FEN: `r4rk15pppp1p5q1p1p3N1P1P1n11P6P1Q2PPP3R1RK1 w - - 0 17`. Identify all skewers in this position. A skewer occurs when a more valuable piece is attacked first and forced to move, exposing a less valuable piece behind it to be captured. For each skewer, provide the key squares in the format: `skewering_piece>front_piece>back_piece` (e.g., `d1>d7>d8`, `a2>e2>h2`). Analyze step by step and explain your reasoning. Finish with a single line formatted EXACTLY as: FINAL ANSWER: `<answer>` If more than one, separate with a comma and a space.

Model response.

Step-by-step reasoning: - A skewer needs three things in a straight line: an attacking line-piece (rook/queen/bishop), a more valuable enemy piece in front, and a less valuable enemy piece behind it, with no blocking pieces between the two enemy pieces along that line. - Check black king alignment: g8 (king) with f8 (rook) behind along the 8th rank would be a classic `K>R` skewer if White could attack from h8 along the rank. However, White has no way to place a rook/queen on the 8th rank from the h-file or otherwise open that line now; the route is blocked and unreachable. - Check a-file: a5 (black queen) is in front of a6 (black pawn), but White cannot line up a rook/queen on the a-file from below because a2 (white pawn) and a4 (white knight) block any rook/queen from the a-file; thus no executable skewer. - Other ranks/files/diagonals: No other pairs of black pieces are aligned with a higher-value piece in front of a lower-value one such that White's rooks/queen can attack along the required line. Black similarly has no current skewer against White (no line-piece is attacking a more valuable white piece with a less valuable white piece behind it). - Conclusion: There are no actual skewers present or immediately playable in this position. FINAL ANSWER: none