VAM: Verbalized Action Masking for Controllable Exploration with Verifiable Rewards

Anonymous ACL submission

Abstract

Reinforcement learning with verifiable rewards (RLVR) has become a practical way to improve large language models (LLMs) when outputs can be checked automatically. Its effectiveness, however, still depends on exploration, since sparse feedback and large action spaces can make rollouts collapse into repeated, uninformative actions. This creates a gap between having a reliable verifier and obtaining useful training signal, since the verifier can score actions while the policy must still discover varied candidates. We propose *Verbalized Action Masking (VAM)*, which uses an explicit action mask to control exploration during LLM post-training with verifiable rewards. VAM samples from a masked action space, verifies mask compliance, and iteratively prunes valid sampled actions from the mask so later rounds explore different candidates before learning from the collected rollouts. We use chess as a controlled, verifiable sequential decision-making domain for studying controllable exploration in RLVR, where legal moves are explicit, action validity is deterministic, move quality can be evaluated automatically, and play requires both short-term tactics and longer-horizon planning. Across Qwen2.5 and Qwen3 models, we evaluate held-out puzzle accuracy and full-game average centipawn loss (ACPL) against GRPO and RFT baselines. These results highlight action masking as a practical mechanism for controllable exploration with verifiable rewards.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) offers a scalable framework for improving large language models (LLMs) (Cobbe et al., 2021; Lightman et al., 2023; OpenAI et al., 2024; Team et al., 2025), where models are post-trained against automatically checkable signals, such as exact correctness or strict output-format compliance. This "verifier-first" framing enables fully automated, on-policy RL loops and has driven strong improvements on reasoning-heavy tasks in recent work (Guo et al., 2025).

RLVR, however, still faces RL's central difficulty of exploration. When the reward is sparse and the effective action space is large, on-policy updates stall unless the policy samples informative behaviors, even with deterministic verification. In many LLM post-training settings with verifiable rewards, this failure mode is easy to miss because the "environment" is often just an evaluator. The learning problem remains that the policy must discover high-reward actions (valid outputs under strict constraints) in an immense space of possible strings.

Classical RL often explores by visiting new states through environment interaction. In contrast, many LLM post-training runs with verifiable rewards repeatedly sample outputs for the same prompt, for example in GRPO (Guo et al., 2025). This makes the central exploration question more local, namely whether the policy tries different plausible actions from the same state or returns the same high-probability action in every rollout group. When a GRPO group collapses to near-identical actions, the group-relative update has little comparative signal, and additional samples largely waste compute.

We propose *Verbalized Action Masking (VAM)* to make action-space exploration controllable. VAM first applies an explicit action mask that restricts the admissible actions for the current state, then learns from the rollouts produced under that mask. A deterministic verifier checks whether the sampled action satisfies the mask; choosing an out-of-mask action receives a fixed penalty and terminates, turning action validity into a verifiable signal suitable for RLVR. Because the mask is verbalized to the model rather than imposed through logits, each rollout remains an on-policy sample from the model conditioned on the current masked action set. We then use the mask to structure exploration via *iterative action-space pruning*. For a fixed state,

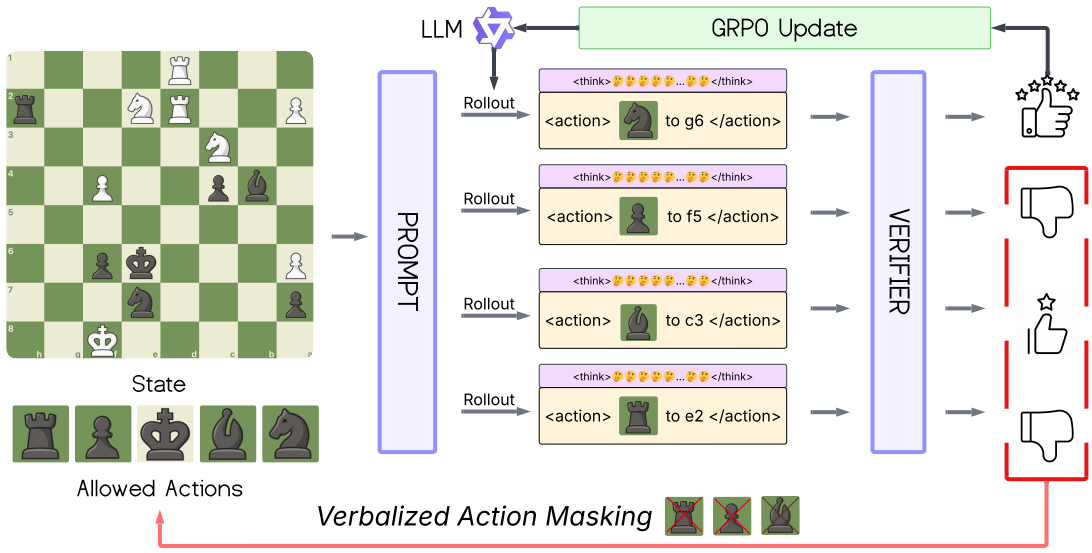

Figure 1: **Overview of VAM with iterative action-space pruning.** For each state, VAM constructs an admissible action mask, verbalizes the current mask to the LLM, parses sampled outputs, and verifies whether each action satisfies the mask before assigning reward. If the current GRPO rollout group misses the target action, VAM removes distinct valid sampled actions from the mask and resamples under the reduced candidate set. This turns repeated GRPO sampling into controlled within-state exploration and produces more informative rollout groups.

we run multiple GRPO sampling rounds, remove already-sampled valid actions from the candidate set, and resample under the pruned set until a target action is found or a fixed budget is exhausted. This mechanism converts repeated sampling into controlled coverage of the action space, since later valid in-mask samples must come from previously unsampled actions.

We use chess as a controlled, verifiable sequential decision-making domain for studying controllable exploration in RLVR. Chess is scientifically useful for this purpose because it is deterministic and fully specified, legal actions are explicit, move quality can be evaluated automatically, and strong play requires both short-term tactical calculation and longer-horizon strategic planning. The method requires an enumerable admissible action set and deterministic checks of mask membership rather than chess-specific search. Recent work similarly uses chess to study LLM reasoning, planning, and policy learning, including reasoning with tactics and explanations (Wang et al., 2025), full-game textual chess modeling and engine-based evaluation (Zhang et al., 2025c), chess game data as a bridge between policy learning and language modeling (Feng et al., 2023), and amortized planning with large-scale transformers (Ruoss et al., 2024a). Prior chess RLVR and LLM-chess studies further show that legal-move lists, structured outputs, and instruction-following constraints materially affect learning and evaluation (Hwang et al., 2025; Kolasani et al., 2025). We therefore apply VAM to chess move selection and evaluate both one-step move selection on held-out puzzles and full-game play against an engine opponent using average centipawn loss (ACPL).

Our contributions are as follows.

1. We formulate an action-masking MDP for RLVR in which each state carries an explicit admissible-action mask, in-mask actions receive verifier rewards, and out-of-mask actions terminate with a fixed penalty;

2. We introduce *Verbalized Action Masking (VAM)*, which verbalizes the current masked action space to an LLM policy, verifies mask membership, and uses iterative action-space pruning so later valid samples explore previously unsampled actions under GRPO-style post-training;

3. We evaluate VAM in text-only chess move selection, a hard, controlled, verifiable sequential decision-making domain. Empirical results on the Qwen2.5 and Qwen3 series models show that VAM improves one-step puzzle accuracy and full-game ACPL relative to GRPO and RFT baselines under the same strict move-selection interface. We also compare fixed-dataset and engine-play training

regimes to characterize the trade-off between puzzle accuracy and full-game move quality.

2 Related Work

Reinforcement learning with verifiable rewards RLVR post-trains LLMs against automatically checkable signals, such as exact correctness, execution feedback, or strict output format compliance. Code-generation RL uses execution feedback as a verifier signal (Le et al., 2022; Yang et al., 2023), while DeepSeek-R1 (Guo et al., 2025) uses GRPO-style RLVR with rule-based correctness and format rewards. Our setting follows the same verifier-first principle, while the checked object is an action selected from an admissible set, with deterministic checks for parsed actions, mask membership, and task rewards.

Exploration in RLVR Verifiable rewards leave useful exploration unresolved when successful or varied rollouts are rare, because GRPO-style updates can still lose diversity. Prior work addresses this bottleneck by changing the reward, data, or optimizer. Outcome-based exploration adds final-answer bonuses or repetition penalties (Song et al., 2025); trial-and-error trajectory optimization collects environment interaction traces and learns from higher-reward trajectories (Song et al., 2024); and entropy or KL-control methods analyze and adjust policy-update dynamics (Deng et al., 2025; Park et al., 2025; Liu et al., 2025). Other methods inject guidance through partial rationales, solution prefixes, expert anchors, or off-policy traces to make rare successes easier to sample (Zhang et al., 2025a; Amani et al., 2025; Qu et al., 2026; Zhang et al., 2025b; Yan et al., 2025), while self-improvement methods iteratively reuse model-generated reasoning or feedback (Zelikman et al., 2022; Madaan et al., 2023). At the action-space level, VAM targets the same exploration failure by changing which valid actions remain available across rollout rounds, rather than by altering the reward, optimizer, or supervision.

Chess and LLMs Chess has been used as a controlled domain for studying LLM reasoning, planning, and policy learning. Wang et al. (2025) study chess as a setting for evaluating reasoning over strategy, tactics, and explanations, and Ruoss et al. (2024a) use chess as a planning domain for evaluating large-scale transformers. ChessGPT (Feng et al., 2023) connects chess game data with policy learning and language modeling, while Chess-LLM (Zhang et al., 2025c) emphasizes textual chess representations, full-game data, and engine-based evaluation. Closer to our setting, Hwang et al. (2025) train LLMs on chess puzzles with GRPO and report that explicit legal-move lists are important for learning, and LLM CHESS (Kolasani et al., 2025) shows that instruction-following and action-space constraints strongly affect chess-play outcomes. In our work, chess serves as a controlled, verifiable sequential decision-making domain for studying controllable exploration in RLVR, while VAM is framed as a general action-space control method for RLVR settings with enumerable actions and deterministic verification.

3 Preliminaries

Chess move-selection notation We represent chess positions as Forsyth–Edwards Notation (FEN) strings and actions as strict lowercase Universal Chess Interface (UCI) moves, such as `e2e4` or `g1f3`. For a position s, let $\mathcal{A}(s)$ denote the legal move set. A model output is valid only if it parses to exactly one strict-UCI move that is legal in s and, for candidate-restricted prompts, belongs to the provided candidate set. We mention Standard Algebraic Notation (SAN) only when contrasting prior work. Because SAN annotations such as `+` for check and `#` for checkmate can serve as additional move-quality cues, our experiments use strict UCI. For full-game evaluation, we report average centipawn loss (ACPL) as an engine-based move-quality metric.

Group Relative Policy Optimization (GRPO) We train with Group Relative Policy Optimization (GRPO) (Guo et al., 2025), which samples a group of G outputs $\{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(O \mid q)$ for each prompt q and computes group-relative advantages

$$A_i = \frac{r_i - \text{mean}(\{r_1, \ldots, r_G\})}{\text{std}(\{r_1, \ldots, r_G\})}, \tag{1}$$

where r_i is the verifier reward for o_i. The clipped GRPO objective can be written compactly as

$$J_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\Big(\min\Big\{\rho_i A_i, \rho_i^{\text{clip}} A_i\Big\} - \beta\, D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\Big)\right], \tag{2}$$

where $\rho_i = \pi_\theta(o_i \mid q)/\pi_{\theta_{\text{old}}}(o_i \mid q)$, $\rho_i^{\text{clip}} = \text{clip}_{1-\epsilon}^{1+\epsilon}(\rho_i)$, and ϵ is the clipping parameter. β

weights the KL penalty. When a group repeats the same action or receives the same reward, the resulting advantages provide little comparative signal. This motivates VAM's focus on action-space exploration within a state.

4 Verbalized Action Masking

Action-space exploration is a central challenge in reinforcement learning with verifiable rewards for LLM post-training. In GRPO (Shao et al., 2024), learning depends on reward differences among multiple sampled outputs for the same prompt. When the per-state action space is large, those samples can collapse to a small set of high-probability actions and yield weak learning signal. VAM addresses this failure mode by making the available action space itself a controllable part of exploration. We first model the RL post-training problem with action masking as an MDP, then introduce VAM and iterative pruning as mechanisms for within-state exploration. This separation keeps the method general because the MDP defines the interface for action availability and rewards, while §5 studies that interface in chess, a demanding domain with explicit legal actions and automatic move-quality evaluation.

4.1 Action-masking MDP

We model the RL post-training problem with action masking as an MDP. An action mask specifies which actions are currently admissible at a state; VAM then controls exploration by changing this mask across sampling rounds. If the agent selects an action outside the current mask, the environment treats it as a constraint violation, assigns a fixed penalty, and terminates the rollout.

Formally, consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, transition kernel $P(s' \mid s, a)$, reward function $R(s, a)$, and discount factor $\gamma \in [0, 1)$. Let $\mathbf{1}[\cdot]$ denote the indicator function. We assume that at each state s the environment can enumerate a finite admissible action set $\mathcal{A}(s) \subseteq \mathcal{A}$. An action mask is a subset $\mathcal{M} \subseteq \mathcal{A}(s)$ of actions that are currently allowed. An *action-masking MDP* augments the state with an explicit mask variable. Specifically, define an augmented state space

$$\tilde{\mathcal{S}} \;=\; \{(s, \mathcal{M}) : s \in \mathcal{S},\ \mathcal{M} \subseteq \mathcal{A}(s)\}, \quad (3)$$

and an action-masking MDP $(\tilde{\mathcal{S}}, \mathcal{A}, \tilde{P}, \tilde{R}, \gamma)$. The mask may depend on the underlying state and can be updated as a function of the chosen action; we write $U(\mathcal{M}' \mid s, a, \mathcal{M})$ for a (possibly deterministic) mask-update kernel. Writing $\tilde{s} = (s, \mathcal{M})$ and $\tilde{s}' = (s', \mathcal{M}')$, for in-mask actions the augmented transition factorizes as

$$\tilde{P}(\tilde{s}' \mid \tilde{s}, a) = P(s' \mid s, a)\, U(\mathcal{M}' \mid s, a, \mathcal{M}). \quad (4)$$

If $a \notin \mathcal{M}$, we treat the transition as terminal (equivalently, a transition to an absorbing terminal state), so the next-state dynamics and mask update are not applied.

The augmented reward assigns the base reward for in-mask actions and a fixed penalty otherwise.

$$\tilde{R}\big((s, \mathcal{M}), a\big) = \begin{cases} R(s, a), & a \in \mathcal{M}, \\ -\mathrm{Penalty}, & a \notin \mathcal{M}, \end{cases} \quad (5)$$

where $\mathrm{Penalty} > 0$ is a constant. We terminate immediately when $a \notin \mathcal{M}$, equivalently with done indicator $d(s, a, \mathcal{M}) = \mathbf{1}[a \notin \mathcal{M}]$. The penalty makes mask violations part of the RL objective while retaining a scalar reward for every sampled output.

4.2 Mask Verbalization for LLM Policies

The action-masking MDP specifies how a mask affects reward and termination. For LLMs, the key design choice is how the policy observes that mask. For LLM policies, we represent the mask directly in the prompt. Each prompt contains a description of the state s and an explicit action mask $\mathcal{M} \subseteq \mathcal{A}(s)$. The model then outputs a single action, and a deterministic parser and verifier map the output to a predicted action $\hat{a}$ and check membership in $\mathcal{M}$. If parsing fails or if $\hat{a} \notin \mathcal{M}$, we treat the outcome as an out-of-mask action, triggering termination and the penalty as defined by the action-masking MDP.

VAM uses task text to expose the current action mask to the policy in a form that standard LLM sampling can use. This text-level representation makes the same action-space control variable visible to the policy, reward function, and pruning loop. The mask is part of the observation, and we sample from the current policy conditioned on that masked prompt, without applying logit-level masking or post-hoc filtering. As a result, each sampled group is on-policy for its corresponding mask-conditioned prompt, which makes the approach compatible with standard GRPO-style objectives.

Algorithm 1 VAM training with iterative action-space pruning.

Procedure PRUNEANDSAMPLE$(\pi, s, \mathcal{M}_0, a^\star)$
Parameters: group size G, max rounds $R_{\max}$, reward $\tilde{R}$
$\mathcal{M} \leftarrow \mathcal{M}_0,\ \mathcal{D}_s \leftarrow \emptyset$
for $r = 1$ **to** $R_{\max}$ **do**
  Sample $\{o_i\}_{i=1}^{G} \sim \pi(\cdot \mid s, \mathcal{M})$ and parse to $\{a_i\}_{i=1}^{G}$
  Compute $r_i \leftarrow \tilde{R}((s, \mathcal{M}), a_i)$ for $i = 1, \ldots, G$
  $\mathcal{D}_s \leftarrow \mathcal{D}_s \cup \{(\mathcal{M}, \{(o_i, r_i)\}_{i=1}^{G})\}$
  $V \leftarrow \{a_i : a_i \in \mathcal{M}\}$
  if $a^\star \in V$ **then**
    break
  $\mathcal{M} \leftarrow \mathcal{M} \setminus V$
  if $\mathcal{M} = \emptyset$ **then**
    break
end for
return $\mathcal{D}_s$
Procedure TRAINVAM$(\pi, \mathcal{D}, T, B)$
Inputs: policy π, dataset $\mathcal{D}$, total steps T, batch size B
for $t = 1$ **to** T **do**
  Sample $\{s_j\}_{j=1}^{B}$ from $\mathcal{D}$ and set $\mathcal{B} \leftarrow \emptyset$
  for $j = 1$ **to** B **do**
    Read $\mathcal{M}_{0,j} \subseteq \mathcal{A}(s_j)$
    Read target $a_j^\star$ if supplied; otherwise set $a_j^\star \in \arg\max_{a \in \mathcal{M}_{0,j}} \mu(s_j, a)$
    $\mathcal{D}_{s_j} \leftarrow$ PRUNEANDSAMPLE$(\pi, s_j, \mathcal{M}_{0,j}, a_j^\star)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{D}_{s_j}$
  end for
  Update π with GRPO on $\mathcal{B}$ (§3)
end for
return π

4.3 Iterative action-space pruning

A static action mask defines which actions are available while leaving within-state diversity to ordinary sampling. VAM therefore adds iterative pruning, in which valid sampled actions are removed from the mask after each rollout group so the next group must place probability mass on the remaining action space.

Verifier-based rewards and target actions In RLVR, an external verifier maps a parsed action to a scalar score. We assume that for each state s and admissible action $a \in \mathcal{A}(s)$, the verifier provides

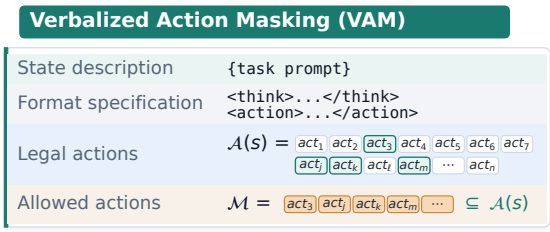

Figure 2: **VAM prompt schema.** A task provides a state description, output contract, legal action set, and current allowed subset. The prompt verbalizes the mask, while deterministic parsing and mask-membership verification connect sampled outputs to the pruning loop in Algorithm 1. For chess, the generic `<action>` field is specialized to `<uci_move>`.

$\mu(s, a) \in \mathbb{R}$, which may be binary or graded. For in-mask actions, we set $R(s, a) = \mu(s, a)$. The parser maps each raw output string o either to a single action $\hat{a} \in \mathcal{A}(s)$ or to an invalid sentinel. If parsing fails, or if $\hat{a} \notin \mathcal{M}$, the sample receives $-\text{Penalty}$ and terminates as a mask violation.

The pruning loop also needs a target action $a^\star$ used only for early stopping. When the task supplies a labeled target, we use it directly; otherwise, we define the verifier-best target within a mask as

$$a^\star(s, \mathcal{M}) \in \arg\max_{a \in \mathcal{M}} \mu(s, a), \tag{6}$$

using a deterministic tie-break when multiple actions share the maximum score. The loop assumes $a^\star \in \mathcal{M}_0$ for the initial mask; if an externally supplied target falls outside the initial mask, that training state must be skipped or the mask must be expanded before pruning.

For each base state s, pruning starts from an initial mask $\mathcal{M}_0 \subseteq \mathcal{A}(s)$. Each round uses a new prompt that differs only in the masked action list, and samples a rollout group of G candidates, matching the GRPO group size (§3). If the target action is absent, we remove the sampled in-mask actions from the candidate set and resample, up to a maximum number of rounds $R_{\max}$. The pruning step is the substantive exploration intervention. Standard GRPO sampling presents the same action set on each retry, whereas VAM changes the mask so valid in-mask actions sampled in later rounds must differ from previously sampled valid actions. The target remains available until it is sampled or the budget is exhausted. If the mask becomes empty, sampling stops for that state. Each pruning round defines a separate mask-conditioned prompt for

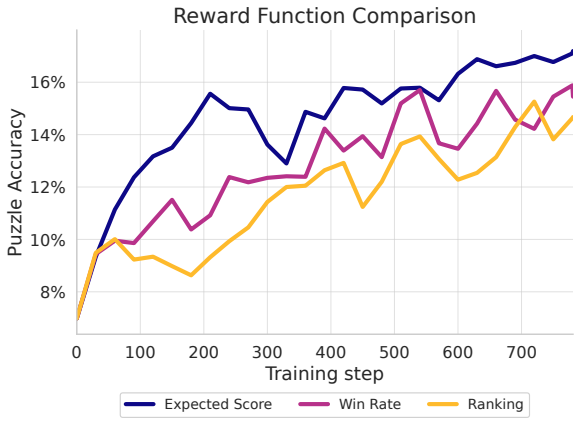

Figure 3: **Reward-function comparison for GRPO.** Puzzle accuracy during fixed-dataset GRPO training for Qwen2.5-7B-Instruct under three engine-derived reward signals: expected score μ_{exp}, win-rate μ_{win}, and rank-based reward μ_{rank}. Expected score performs best among the three reward signals.

GRPO and yields a sequence of contrastive groups for the same underlying state. Algorithm 1 summarizes how these groups are collected and optimized.

4.4 Applicability of VAM

VAM applies when a task exposes enough structure for action-space control to be meaningful. The environment must provide a finite action set for each state, deterministic checks for mask membership, and a verifier that scores in-mask actions. These conditions make chess a useful evaluation domain. In chess, a position determines its legal UCI moves, mask membership is a string-level and rule-level check, and an engine supplies deterministic move-quality scores. The experiments specialize the general interface to this domain while leaving the masking and pruning mechanism unchanged.

5 Experiments

We evaluate VAM in text-only chess move selection, a controlled and verifiable sequential decision-making setting where the model selects a move from a provided list without search or external tools. Chess is challenging because strong play requires choosing among plausible legal moves whose values depend on future play, yet it supplies enumerable actions, strict validity checks, and deterministic engine-derived rewards. We report one-step move selection on Searchless Chess puzzle positions (Ruoss et al., 2024b) and full-game play against a fixed engine opponent, measured by average centipawn loss (ACPL).

5.1 Research Questions

We organize the evaluation around three research questions:

RQ1: Does VAM improve within-state exploration on held-out move-selection puzzles?

RQ2: Do these gains translate to full-game move quality?

RQ3: How does engine-play position generation compare with fixed-dataset training?

5.2 Experimental Setup

Models We run experiments with Qwen2.5 and Qwen3 models. The Qwen2.5 experiments use Qwen2.5-3B-Instruct and Qwen2.5-7B-Instruct (Qwen et al., 2025). The added Qwen3 experiments use Qwen3-1.7B-Base and Qwen3-4B-Base (Yang et al., 2025).

Move-selection task Each decision prompt provides the position as a FEN string, the side to move, and an ordered legal-move list in strict lowercase UCI. VAM additionally provides a candidate subset, `allowed_moves`, which serves as the initial mask $\mathcal{M}_0$. Models output a single move in strict lowercase UCI inside a `<uci_move>` tag, and we evaluate with a deterministic parser and legality checker. We use a prompt template based on Hwang et al. (2025), modified to use strict UCI and, for VAM, an explicit `allowed_moves` mask. The same parser is used for all methods, so differences reflect move selection under the verifier rather than looser validity criteria; the full templates are in Appendix I.

State distributions for training We focus on RLVR methods that learn from deterministic engine-provided signals. Each training position is paired with a per-move value map $\mu(s, a) \in [0, 1]$ over legal moves, used to define rewards (§4.3). We compare two state sources, a fixed 100,000-position Searchless Chess dataset (Ruoss et al., 2024b) aligned with Chess-R1 (Hwang et al., 2025) and an on-policy stream generated via engine-play against a fixed engine opponent (Algorithm 2). Both sources use the same VAM objective in Algorithm 1. Fixed-dataset training isolates the effect of pruning on move-selection exploration, while engine-play position generation tests whether the same action-space control improves learning from states induced by actual play. We also directly compare engine-play and fixed-dataset training for VAM.

Table 1: **Main results across Qwen2.5 and Qwen3 models.** Puzzle performance is accuracy on the Searchless Chess held-out puzzle set (higher is better). ACPL is average centipawn loss against fixed Stockfish opponents, computed by a depth-20 analyzer (lower is better). Bold marks the best value within each model block, following the indicated metric direction.

Training run		Puzzle	Full-game play		
Model	Method	Puzzle acc. ↑ (%)	Overall ACPL ↓	Depth 5 ACPL ↓	Depth 1 ACPL ↓
Qwen2.5-7B-Instruct	VAM	**28.60**	**75.72**	**69.68**	**81.77**
	GRPO	16.32	137.48	117.95	157.00
	RFT	9.34	526.44	509.80	543.08
Qwen2.5-3B-Instruct	VAM	**24.57**	**83.04**	**84.90**	**81.18**
	GRPO	13.32	380.54	377.38	383.71
	RFT	7.02	559.09	503.79	614.38
Qwen3-4B-Base	VAM	24.54	**89.69**	**87.63**	**91.75**
	GRPO	**24.75**	114.17	136.34	92.01
	RFT	9.40	596.47	688.008	504.93
Qwen3-1.7B-Base	VAM	**17.51**	**111.76**	**114.90**	**120.36**
	GRPO	13.96	202.20	217.94	186.47
	RFT	11.30	460.86	467.95	453.77

Puzzle benchmark We evaluate one-step move selection on the Lichess puzzle positions in Searchless Chess (Ruoss et al., 2024b). Each position provides a FEN string, an ordered UCI legal-move list, and a provided puzzle solution move. We use all 10,000 positions as a held-out test set for puzzle evaluation. We treat the provided solution move as the target action for puzzle accuracy.

Full-game benchmark We evaluate end-to-end play by playing complete games from the standard initial position against a fixed Stockfish opponent (Skill Level 0), reported separately for depth 1 and depth 5. We use a fixed protocol for game termination, invalid-move retries, and ACPL computation, recorded in Appendix H.2.

Metrics For puzzles, we report accuracy for selecting the provided puzzle solution under the candidate set. For full games, we report ACPL computed by a separate Stockfish analyzer that runs at depth 20 and serves as an expert evaluator. Appendix H.2 records the game-level and move-level aggregation rules.

5.3 Baselines

We compare VAM against GRPO baselines that use the same engine verifier and strict UCI output contract and omit iterative pruning. This setting follows Chess-R1 (Hwang et al., 2025), with the modification that we use strict UCI (instead of SAN) to avoid SAN annotations, such as + for check and # for checkmate, that can act as additional tactical cues. To probe how reward design affects exploration under GRPO, we evaluate three reward definitions derived from the engine signal, namely expected score, win rate, and ranking among moves. We include a Rejection Sampling Fine-tuning (RFT) baseline (Yuan et al., 2023) trained on the same fixed dataset as a supervised reference point. Additional training, verifier, and optimization details are given in Appendices A–F.

Reward definitions from the engine signal. For each position s and legal move $a \in \mathcal{A}(s)$, Stockfish (The Stockfish Developers, 2024) provides an evaluation that we convert into a scalar verifier score $\mu(s,a) \in [0,1]$. In our main experiments we use the following win/-draw/loss (WDL) based expected-score proxy. If the engine reports win/draw/loss likelihoods $\big(p_W(s,a), p_D(s,a), p_L(s,a)\big)$ (with $p_W + p_D + p_L = 1$), we define

$$\mu_{\text{exp}}(s,a) \;=\; p_W(s,a) + \tfrac{1}{2}p_D(s,a). \qquad (7)$$

We consider two additional reward variants for the GRPO baselines. *(i) win-rate reward* $\mu_{\text{win}}(s,a) = p_W(s,a)$, and *(ii) rank-based reward* computed from the within-position ordering of $\mu_{\text{exp}}(s,a)$ as

$$\mu_{\text{rank}}(s,a) \;=\; 1 - \frac{\text{rank}_{\downarrow}\big(\mu_{\text{exp}}(s,a)\big) - 1}{|\mathcal{A}(s)| - 1}, \qquad (8)$$

where $\text{rank}_{\downarrow}$ assigns rank 1 to the highest-scoring move (ties broken deterministically). Unless stated otherwise, VAM and GRPO baselines use μ_{exp}.

5.4 Results

Main results Table 1 summarizes puzzle accuracy and full-game ACPL for all evaluated models and methods. The main trend is that VAM is the strongest overall method across the evaluation suite. It gives the lowest full-game ACPL for every model and opponent depth, and it gives the best puzzle accuracy for three of four models while closely matching GRPO on the remaining Qwen3-4B run. This supports the central hypothesis that iterative pruning makes grouped RL rollouts more useful by forcing coverage over distinct valid moves. GRPO uses the same verifier and strict output contract but samples from an unchanged action set, while RFT provides a supervised reference without on-policy verifier feedback. The shared protocol therefore isolates iterative pruning as the main difference between VAM and the GRPO baseline.

Within-state exploration on puzzles Figure 3 studies the reward signal used for fixed-dataset GRPO. The expected-score signal gives the strongest GRPO curve among the reward variants we tested, so subsequent comparisons use expected score for both GRPO baselines and VAM. Under this reward, Table 1 shows that VAM improves held-out puzzle accuracy over RFT for every model and over GRPO for all but the Qwen3-4B run, where the two RL methods are nearly tied. The puzzle results are consistent with the intended within-state exploration mechanism: after frequent valid moves are sampled, VAM removes them from later rollout rounds, making additional valid samples cover new legal candidates rather than repeating the same high-probability actions.

Full-game move quality Table 1 also reports full-game move quality. VAM also leads on ACPL, including the depth-specific opponent columns, indicating that better action-space exploration does not only improve one-step puzzle selection but also carries over to complete games, and that it holds across model sizes and opponent depths.

Fixed-dataset versus engine-play training Beyond the main comparison across models and methods, Figure 4 isolates the training regime for VAM. We ask whether VAM's behavior changes when the training positions come from a fixed dataset or from engine-play. We study this question on the Qwen2.5 models, where both training regimes are available. Figure 4 shows a consistent trade-off: fixed-dataset training gives stronger one-step puzzle accuracy, while engine-play position generation gives better full-game ACPL.

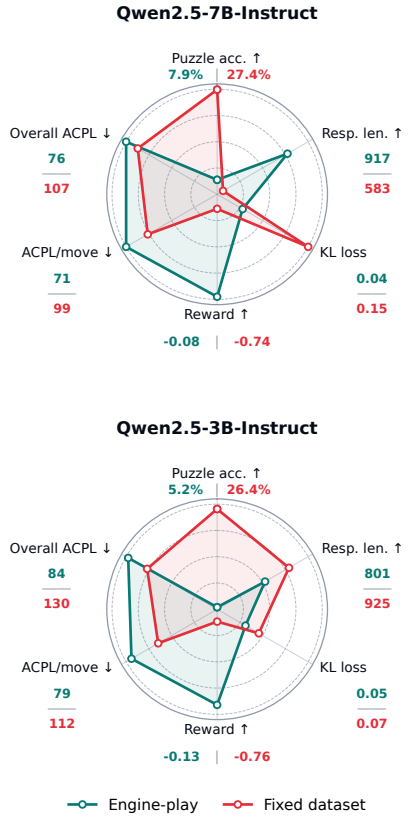

Figure 4: **Training-regime comparison for VAM.** Radar plots compare fixed-dataset and engine-play training for the 7B and 3B models across puzzle accuracy, response length, KL loss, verifier reward, ACPL per move, and overall ACPL. Raw values are annotated; lower ACPL is better.

6 Conclusion

We introduced Verbalized Action Masking (VAM), an action-masking approach for RLVR that improves exploration by removing sampled valid actions from later masks within the same state. We evaluated VAM in text-only chess move selection with strict UCI outputs, using one-step puzzle selection and full-game play against an engine opponent. Across Qwen2.5 and Qwen3 models, VAM is compared with GRPO and RFT under the same move interface and verifier-backed metrics; expected score is the strongest reward choice for GRPO baselines on puzzles. The fixed-dataset and engine-play results show complementary strengths, suggesting that the training state distribution is a key design choice for LLM post-training in structured action spaces with verifiable rewards.

7 Limitations

VAM assumes that each state exposes an enumerable action set, deterministic mask-membership checks, and a verifier that can score in-mask actions. These assumptions hold naturally in chess. Open-ended generation tasks and domains with very large action spaces would require candidate generation, approximate verification, or another mechanism for constructing manageable masks. Our experiments use text-only chess move selection with strict UCI outputs and engine-derived rewards, so the evidence is confined to structured, verifiable action spaces with finite actions.

References

Mohammad Hossein Amani, Aryo Lotfi, Nicolas Mario Baldwin, Samy Bengio, Mehrdad Farajtabar, Emmanuel Abbe, and Robert West. 2025. Rl for reasoning by adaptively revealing rationales. *arXiv preprint arXiv:2506.18110*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jia Deng, Jie Chen, Zhipeng Chen, Daixuan Cheng, Fei Bai, Beichen Zhang, Yinqian Min, Yanzipeng Gao, Wayne Xin Zhao, and Ji-Rong Wen. 2025. From trial-and-error to improvement: A systematic analysis of llm exploration mechanisms in rlvr. *arXiv preprint arXiv:2508.07534*.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dongyoon Hwang, Hojoon Lee, Jaegul Choo, Dongmin Park, and Jongho Park. 2025. Can large language models develop strategic reasoning? post-training insights from learning chess. *arXiv preprint arXiv:2507.00726*.

Sai Kolasani, Maxim Saplin, Nicholas Crispino, Kyle Montgomery, Jared Quincy Davis, Matei Zaharia, Chi Wang, and Chenguang Wang. 2025. Llm chess: Benchmarking reasoning and instruction-following in llms through chess. *Preprint*, arXiv:2512.01992.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

OpenAI, :, Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, Alex Iftimie, Alex Karpenko, Alex Tachard Passos, Alexander Neitz, Alexander Prokofiev, Alexander Wei, Allison Tam, and 244 others. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Jaesung R Park, Junsu Kim, Gyeongman Kim, Jinyoung Jo, Sean Choi, Jaewoong Cho, and Ernest K Ryu. 2025. Clip-low increases entropy and clip-high decreases entropy in reinforcement learning of large language models. *arXiv preprint arXiv:2509.26114*.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. 2026. Pope: Learning to reason on hard problems via privileged on-policy exploration. *arXiv preprint arXiv:2601.18779*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li K Wenliang, Elliot Catt, John Reid, Cannada A Lewis, Joel Veness, and Tim Genewein. 2024a. Amortized planning with large-scale transformers: A case study on chess. *Advances in Neural Information Processing Systems*, 37:65765–65790.

Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, and Tim Genewein. 2024b. Grandmaster-level chess without search. *arXiv preprint arXiv:2402.04494*, 2.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024. Trial and error: Exploration-based trajectory optimization for llm agents. *arXiv preprint arXiv:2403.02502*.

Yuda Song, Julia Kempe, and Remi Munos. 2025. Outcome-based exploration for llm reasoning. *arXiv preprint arXiv:2509.06941*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

The Stockfish Developers. 2024. Stockfish. Source code available at https://github.com/official-stockfish/Stockfish.

Shu Wang, Lei Ji, Renxi Wang, Wenxiao Zhao, Haokun Liu, Yifan Hou, and Ying Nian Wu. 2025. Explore the reasoning capability of llms in the chess testbed. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 611–622.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. 2025. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *Advances in Neural Information Processing Systems*, 36:23826–23854.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Kaiyi Zhang, Ang Lv, Jinpeng Li, Yongbo Wang, Feng Wang, Haoyuan Hu, and Rui Yan. 2025a. Stephint: Multi-level stepwise hints enhance reinforcement learning to reason. *arXiv preprint arXiv:2507.02841*.

Xuechen Zhang, Zijian Huang, Yingcong Li, Chenshun Ni, Jiasi Chen, and Samet Oymak. 2025b. Bread: Branched rollouts from expert anchors bridge sft & rl for reasoning. *arXiv preprint arXiv:2506.17211*.

Yinqi Zhang, Xintian Han, Haolong Li, Kedi Chen, and Shaohui Lin. 2025c. Complete chess games enable llm become a chess master. *arXiv preprint arXiv:2501.17186*.

A Training Data and Position Collection

A.1 Training Position Sources

We consider two sources of training positions. First, a fixed 100,000-position dataset from Searchless Chess (Ruoss et al., 2024b), aligned with the Chess-R1 setup (Hwang et al., 2025). Second, an on-policy dataset generated via engine-play against an engine opponent, where we record encountered positions whenever it is the model's turn and query the engine to obtain $\mu(s, a)$ over the legal move list (Algorithm 2).

Artifact licenses The fixed Searchless Chess dataset is released under the Apache License and Creative Commons Attribution 4.0 (CC-BY), and Stockfish (The Stockfish Developers, 2024) is licensed under the GNU General Public License v3.0. Our use is consistent with these artifacts' intended use: the dataset supports chess move-selection training and evaluation, and Stockfish is used as a chess engine and analyzer for move-quality signals.

Algorithm 2 Engine-play position collection

Procedure COLLECTBY-PLAY($\pi, E, \text{Verifier}, B$)
Inputs: policy π, environment E with a fixed engine opponent, target data size B, pool size N
Initialize or reuse $\mathcal{E} = \{e_1, \ldots, e_N\}$ from E, and set $\mathcal{D}_{\text{play}} \leftarrow \emptyset$
while $|\mathcal{D}_{\text{play}}| < B$ **do**
  Retrieve current model-to-move states $\{s_i\}_{i=1}^{M}$ from $\mathcal{E}$
  for $i = 1$ **to** M **do**
    Read $\mathcal{A}(s_i)$ and query Verifier for $\mu(s_i, a)$ for all $a \in \mathcal{A}(s_i)$
    Append $(s_i, \mathcal{A}(s_i), \mu(s_i, \cdot))$ to $\mathcal{D}_{\text{play}}$
    if $|\mathcal{D}_{\text{play}}| \geq B$ **then**
      break
  end for
  Step all instances in $\mathcal{E}$ with π on model turns and the fixed engine on opponent turns
end while
return $\mathcal{D}_{\text{play}}$

A.2 Fields Per Training Position

Each training example includes the position (FEN), an ordered legal-move list in strict UCI, and an engine-provided per-move value map $\mu(s, a)$ used for reward computation. For the fixed dataset, each example also includes a dataset-provided solution move in UCI. For VAM-style selection prompts, we additionally specify an ordered candidate subset `allowed_moves` that serves as the initial mask $\mathcal{M}_0$; when a candidate subset is not provided by the dataset, $\mathcal{M}_0$ can be set to the full legal set $\mathcal{A}(s)$ (§4.3).

B Training Code

Anonymous code repository: `https://anonymous.4open.science/r/arr-vam-6552`. It contains the script for reproducing the reported experiments. Our implementation builds on the open-source veRL framework (Sheng et al., 2024) (`verl-project/verl`), which is licensed under Apache-2.0.

C Compute Resources

Training used one or two 4 × NVIDIA GH200 Grace Hopper Superchip nodes, each with 4 Grace CPUs (72 cores and 120 GB memory each), 4 H100 Tensor Core GPUs (96 GB memory each), and NVIDIA NVLink-C2C interconnect. The longest runtime was around 12 hours on two nodes.

D Potential Risks

This paper presents work whose goal is to advance the field of machine learning, specifically methods for reinforcement learning post-training of large language models under verifiable constraints. There are many potential societal consequences of improved LLM capabilities and of broader adoption of RL post-training, none of which we feel must be specifically highlighted here.

E Use of AI Assistants

We used AI assistants to help with manuscript writing, such as grammar correction and wording edits.

F Optimization and Verifier Details

F.1 Rejection Sampling Fine-Tuning (RFT) Baseline

We construct the supervised baseline dataset from the same fixed 100,000-position dataset described in §A. Each example provides a dataset-provided solution move in UCI. We use the baseline prompt template without a candidate subset and apply rejection sampling over model generations, keeping only generations whose parsed strict UCI move matches the dataset-provided solution (rejecting

malformed or illegal outputs). This procedure yields 45,806 accepted examples for Qwen2.5-3B-Instruct (5.73% acceptance) and 71,181 accepted examples for Qwen2.5-7B-Instruct (8.90% acceptance). For Qwen3, the same rejection-sampling procedure yields 43,168 accepted examples for Qwen3-1.7B-Base and 60,064 accepted examples for Qwen3-4B-Base. Table 2 lists the settings used for this baseline.

F.2 Engine-Derived Verifier Scores

Stockfish-based scoring We compute the verifier score $\mu(s, a)$ using Stockfish (The Stockfish Developers, 2024). In the main experiments, we use the expected-score proxy μ_{exp} (Eq. 7) derived from the engine's win/draw/loss probabilities, and define the target action within a candidate set as the μ-best move with a deterministic tie-break (§4.3). Section 5.3 defines additional reward variants (μ_{win} and μ_{rank}) used in the GRPO reward-function comparison.

Reward assignment under the candidate mask For both VAM and GRPO baselines, the reward is computed from the decoded move and the engine metadata for the current position. If the decoded move is well-formed, legal, and (for selection prompts) inside `allowed_moves`, it receives the engine-derived score $\mu(s, a)$. Otherwise (missing tag, malformed UCI, illegal move, or out-of-candidate move), we treat the output as invalid and assign a fixed negative penalty, terminating the rollout (§4).

F.3 Hyperparameters

Table 2 summarizes the main hyperparameters used across experiments. Shared optimization settings are matched across RL methods for fair comparisons, and VAM-specific controls are listed in the VAM column. The last two columns report the RFT baseline settings.

Training steps The Qwen2.5 experiments use 600 training steps, and the Qwen3 experiments use 400 training steps.

G Prompting and Parsing Protocol

Inputs per decision Each decision includes the FEN string, an explicit side-to-move field, and an ordered list of legal moves in strict lowercase UCI. In the candidate-restricted (selection) setting we additionally provide an ordered subset `allowed_moves` (also called `considered_moves_uci`) and require the model to choose from that subset.

Required output The model must output a single move inside a `<uci_move>` tag, with a strict lowercase UCI string as payload (for example, `e2e4`). Promotions include the piece letter (for example, `e7e8q`). Castling is represented in UCI as the king move (for example, `e1g1` for white kingside castling).

Validity checks We treat any unparsable output, illegal move, or (for selection prompts) out-of-candidate move as invalid. For selection prompts, the decoded move must match one of the provided candidate strings exactly.

H Evaluation Protocols

H.1 Puzzle Evaluation Protocol

Data fields Each puzzle row provides `fen`, an ordered `legal_moves_uci` list, and a provided solution move in UCI. Some rows additionally include per-move value maps defined over the legal move list, stored as JSON keyed by UCI strings. The value maps include an engine-derived expected score or win-probability proxy in $[0, 1]$.

Targets and metrics We treat the dataset-provided solution move as the target action $a^\star$ for evaluation. The main paper reports accuracy (success if $\hat{a} = a^\star$), where invalid outputs count as failures. We additionally track format and legality compliance and, when value maps are available, the mean selected-move value $\mathbb{E}[\mu(s, \hat{a})]$.

H.2 Full-Game Evaluation Protocol

Match protocol Games start from the standard initial position and are capped at 200 plies. We do not automatically claim draws. Unless stated otherwise, we play 50 games per evaluation depth (depth 1 and depth 5), with per-depth color balancing and a deterministic shuffle.

Invalid moves If the model output is unparsable or illegal, we retry up to three attempts. Persistent failure is treated as resignation or forfeit, which affects win rate and can yield worst-case ACPL when no valid moves are played.

ACPL computation We compute centipawn loss using a separate Stockfish analyzer at depth 20, with approximately 1 second of analysis per move.

Table 2: Hyperparameters for GRPO, VAM, and RFT.

Parameter	GRPO	VAM	Parameter	RFT
Batch size	128	128	Global batch size	32
Mini batch size	128	128	Total source prompts	100,000
GRPO group size	8	8	Samples per prompt	8
Sampling temperature	1.0	1.0	Sampling temperature	0.6
Sampling top-p	1.0	1.0	Sampling top-p	0.9
Max response tokens	2000	2000	Max response tokens	2000
Epochs	1	1	Epochs	2
Optimizer	AdamW	AdamW	Optimizer	AdamW
Learning rate	1e-6	1e-6	Base learning rate	1e-5
LR schedule	constant	constant	LR schedule	cosine
Warmup ratio	10 steps	10 steps	Warmup ratio	0.1
Weight decay	0.01	0.01	Weight decay	0.01
Gradient clipping	1	1	Gradient clipping	1
KL loss coefficient	1e-3	1e-3	#RFT examples	Qwen2.5-3B: 45,806
Entropy coefficient	0	0		Qwen2.5-7B: 71,181
Clip ratio	0.2	0.2		Qwen3-1.7B: 43,168
Reward function	$\mu_{\text{exp}}/\mu_{\text{win}}/\mu_{\text{rank}}$	μ_{exp}		Qwen3-4B: 60,064
$R_{\max}$	/	4		
Online opponent	/	stockfish (skill 0, depth 1)		

Mate scores are mapped to 1000, and centipawn values are clamped to a fixed cap, following the evaluation protocol used for the reported full-game results. Per-move centipawn loss (CPL) uses the before versus after evaluations, with evaluations converted to the mover's point of view and clipped at 0:

$$\text{CPL} = \max\big(0, E_{\text{before}} - E_{\text{after}}\big). \tag{9}$$

We first compute per-game ACPL as the mean CPL over the model's moves within that game. We then report two aggregates across a batch of games: (i) *overall ACPL*, the mean of per-game ACPL values (each game equally weighted), and (ii) *ACPL per move*, a move-weighted mean over all model moves in the batch.

I Prompt Templates

We include the exact prompt templates used for the Qwen3 base-model experiments in Figures 5 and 6. We use `<reason>` rather than `<think>` because Qwen already uses `<think>`, and the base model did not reliably follow instructions to emit reasoning enclosed in `<think>` tags.

VAM prompt template

```
A conversation between User and Assistant.
The User gives a chess position, legal UCI
moves, and allowed_moves. The Assistant
chooses one move from allowed_moves.

The Assistant is a strong chess player. The
Assistant analyzes the position, compares
candidate moves, considers the opponent's
best replies, and chooses the best move for
the side to move.

The Assistant writes the reasoning inside
<reason>...</reason>, then writes the
selected move inside
<uci_move>...</uci_move>. The selected move
must be copied exactly from allowed_moves.

Rules:
Choose exactly one move from allowed_moves.
Use allowed_moves as the candidate set, not
as a ranking or quality label. The order of
allowed_moves does not indicate quality. Do
not infer move quality from the order of
allowed_moves or from how allowed_moves was
selected.
If allowed_moves has one move, choose that
move.
Use the legal moves list to understand the
position, but the final move inside
<uci_move> must be copied exactly from
allowed_moves.
Analyze the position by considering
material, king safety, piece activity, pawn
structure, threats, forcing tactics, and
likely opponent replies.
Compare the important moves in allowed_moves
and choose the move that gives the best
expected outcome for the side to move.
A UCI move is source square plus target
square, with optional promotion letter q, r,
b, or n.

Reminder of chess rules:
- Bishops move diagonally.
- Rooks move horizontally or vertically.
- Knights jump in an L-shape.
- Queens combine rook and bishop movement.
- Kings move one square in any direction.
- Pawns move forward, capture diagonally,
and can promote.

After the reasoning, close </reason>, then
write <uci_move>, the chosen UCI move, and
</uci_move>. Do not continue after
</uci_move>.

User:
FEN:
{{ FEN }}

Legal moves:
{{ legal_moves_uci_list | join('\n') }}

allowed_moves:
{{ considered_moves_uci_list | join('\n') }}

Assistant:
<reason>
```

Figure 5: VAM prompt template.

Baseline prompt template

```
A conversation between User and Assistant.
The User gives a chess position and legal
UCI moves. The Assistant chooses one legal
move.

The Assistant is a strong chess player. The
Assistant analyzes the position, compares
candidate moves, considers the opponent's
best replies, and chooses the best legal
move for the side to move.

The Assistant writes the reasoning inside
<reason>...</reason>, then writes the
selected move inside
<uci_move>...</uci_move>. The selected move
must be copied exactly from the legal moves.

Rules:
Choose exactly one legal move for the side
to move.
Compare candidate moves by chess analysis,
including material, king safety, piece
activity, pawn structure, threats, forcing
tactics, and likely opponent replies.
Choose the move that gives the best expected
outcome for the side to move.
The final move inside <uci_move> must be
copied exactly from the legal moves.
A UCI move is source square plus target
square, with optional promotion letter q, r,
b, or n.

Reminder of chess rules:
- Bishops move diagonally.
- Rooks move horizontally or vertically.
- Knights jump in an L-shape.
- Queens combine rook and bishop movement.
- Kings move one square in any direction.
- Pawns move forward, capture diagonally,
and can promote.

After the reasoning, close </reason>, then
write <uci_move>, the chosen UCI move, and
</uci_move>. Do not continue after
</uci_move>.

User:
FEN:
{{ FEN }}

Legal moves:
{{ legal_moves_uci_list | join('\n') }}

Assistant:
<reason>
```

Figure 6: Baseline prompt template.