RIVER: RECURSIVE SELF-IMPROVEMENT OF LLM POKER AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent skills, natural-language instructions written by human experts, are increasingly used to extend large language model (LLM) agents without retraining, and have recently been applied to strategic games such as poker. For a recent expert-written skill for heads-up no-limit Texas hold'em (PokerSkill), we find that the return varies considerably across base models, across reasoning efforts of a single model, and even across requests for more detailed analysis before acting that leave the encoded expertise unchanged. More reasoning thus does not guarantee better play, and the value of a skill depends on how each model executes it. Since we cannot know in advance how a model will apply fixed human expertise, this raises the question of whether a model can start from human expertise and iteratively derive a poker skill suited to itself. Poker makes such self-improvement difficult, since single-hand outcomes are highly variable with no verifiable reward for individual decisions, and the decision space is too large for hand-written advice to cover. We introduce RIVER (**R**ecursive **I**mprovement **v**ia **E**nsemble **R**eflection), in which the LLM that plays poker recursively improves its own skill library. Each update reflects retrospectively on three complementary evidence streams, namely the human expertise encoded in the original skill, the model's execution of its current skill, and a comparative analysis of returns across skill versions. We further factorize the skill by decision factor and aggregate evidence from single decisions to similar states, condensing the reasons behind them into updates of the skill. Across four configurations spanning three model families, RIVER derives for every model a skill that reaches a positive mean return against the Slumbot poker bot and exceeds the expert-written skill by 383 to 526 milli-big-blinds per hand.

1 INTRODUCTION

Large language models (LLMs) bring broad world knowledge to agentic tasks, yet many tasks call for specialized procedural expertise, which agent skills supply as natural-language instructions loaded at inference time (Xu & Yan, 2026). Because skills act through the model's input, they can adapt frontier models served only through APIs (Zhang et al., 2026b; Zhao et al., 2024). Strategic games with imperfect information make such expertise valuable, since strong play rests on principles for handling hidden information, deception, and multi-step planning that general-purpose LLMs struggle to apply on their own (Lin et al., 2026). Heads-up no-limit Texas hold'em is a long-standing benchmark of this kind, where superhuman play has relied on equilibrium solvers with massive computational budgets (Moravčík et al., 2017; Brown & Sandholm, 2018). PokerSkill (Li et al., 2026) shows that an expert-written skill, combined with a deterministic context engine, lets frontier LLMs play this game competitively and improves their returns without training or solver access.

We evaluate the released PokerSkill against the Slumbot poker bot (Jackson, 2013) and find that its return varies considerably across base models, across reasoning efforts of a single model, and even across requests for more detailed analysis before acting that leave the encoded expertise unchanged (Section 2). The effectiveness of a skill is therefore tied to the model and configuration that execute it, consistent with the observation that different models call for different agent designs (Lee et al., 2026). The human expertise in a skill is fixed, yet we cannot know in advance how a given model will apply it, and writing expertise tailored to every model is impractical. This raises the question of whether a model can start from human expertise and iteratively derive a poker skill suited to itself.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Many works (Yang et al., 2024; Agrawal et al., 2026; Zhang et al., 2026b;a; Zelikman et al., 2023) study such self-improvement for prompts, contexts, and agent code, and they typically compare candidate versions with a verifiable signal, such as the correctness of a mathematical answer or the outcome of unit tests. Poker complicates such self-improvement in three ways. First of all, the game is partially observable. The value of a decision depends on the opponent's hidden cards and unknown strategy, and cannot be judged from what the player observes. Moreover, returns have high variance. The return of a hand arrives only after a sequence of decisions, so a sound decision can lose to the opponent's cards, and thousands of hands are needed before a difference in mean return becomes visible (Burch et al., 2018). Finally, the decision space is vast. Each decision depends jointly on hand strength, board texture, the preceding actions, stack depth, position, and street, so hand-written advice cannot cover every situation, and a change in return is hard to trace back to the advice responsible. We therefore need a self-improvement method that learns effectively from noisy, indirect evidence and organizes the skill to cover a vast decision space.

To address these difficulties, we introduce RIVER (**R**ecursive **I**mprovement **v**ia **E**nsemble **R**eflection), in which the model that plays poker recursively improves its own skill library without changing its weights. RIVER consists of three components. A *factorized skill library* decomposes the skill into six factor skills and compares decisions made in similar states, so that each part of the skill has a clear scope. *Ensemble reflection* gathers evidence from three complementary streams, namely the human expertise in PokerSkill, an audit of how the model executes its current skill, and an analysis of returns across versions, which together make up for the missing decision-level label. *Hierarchical induction* aggregates the evidence from single decisions to similar states, analyzes the reasons behind them, and condenses these reasons to update the skill. We apply RIVER to four configurations spanning three model families, and every model recursively improves a skill of its own that exceeds PokerSkill. Our main contributions are summarized below.

- **More reasoning does not guarantee better poker play.** On heads-up no-limit hold'em, raising the reasoning effort or asking for more detailed analysis, while leaving the expert knowledge unchanged, raises the return of some models and lowers that of others. The value of a skill therefore depends on how each model executes it.
- **A recursive self-improvement method for poker.** We propose RIVER, in which the model that plays poker recursively improves its own skill library. Through a factorized skill library, ensemble reflection, and hierarchical induction, RIVER lets the model derive a skill suited to itself from noisy, indirect evidence across a vast decision space.
- **Model-specific poker skills through recursive improvement.** We experiment on four configurations spanning three model families, where each model evolves a skill of its own that exceeds PokerSkill by 383 to 526 milli-big-blinds per hand (mbb/hand).

2 Same Expertise, Different Returns

Setup. We evaluate PokerSkill (Li et al., 2026), a recent expert-written skill for heads-up no-limit hold'em, by letting each model play against the Slumbot poker bot (Jackson, 2013) with the PokerSkill prompt. Following PokerSkill, we report the mean return in milli-big-blinds per hand (mbb/hand), each over 5,000 hands. Appendix E discusses evaluation variance and the limits of available variance-reduction techniques. We vary two conditions. The *reasoning effort* is an API setting of the model, with levels low and medium. The *analysis request* specifies what the model is asked to write before its action. The short format asks for brief reasoning, and the evidence–analysis–decision (EAD) format asks the model to lay out the evidence and analyze it before deciding. The expert guidance in the prompt is identical in both formats. Table 1 collects all results.

The same skill yields different returns across models. With the short format and low reasoning effort, the return ranges from -54.7 mbb/hand for Gemini 3.7 Flash to -1019.0 for GPT-5.6 Sol, with DeepSeek V4 Flash at -266.1 and GPT-6 Luna at -318.8 (Table 1a). At medium effort, GPT-6 Luna and Gemini 3.7 Flash have similar observed returns of -166.6 and -190.4 mbb/hand.

Reasoning effort has model-dependent returns. With the short format, Gemini 3.7 Flash records -54.7 mbb/hand at low effort and -190.4 at medium effort, while GPT-5.6 Sol records -1019.0 and -475.5, respectively (Table 1a). The changes in return vary in direction and size.

Table 1: Mean return (mbb/hand) of the PokerSkill prompt against Slumbot over 5,000 hands per setting. Bold marks the better setting in each row.

(a) Reasoning effort, short format

Model	Low	Medium
Gemini 3.7 Flash	$\mathbf{-54.7}$	-190.4
GPT-6 Luna	-318.8	$\mathbf{-166.6}$
GPT-5.6 Sol	-1019.0	$\mathbf{-475.5}$
DeepSeek V4 Flash	-266.1	$\mathbf{-247.7}$

(b) Analysis request

Model	Effort	Short	EAD
Gemini 3.7 Flash	low	$\mathbf{-54.7}$	-143.7
Gemini 3.7 Flash	medium	$\mathbf{-190.4}$	-301.9
GPT-5.5	medium	-271.7	$\mathbf{-155.5}$
DeepSeek V4 Flash	low	$\mathbf{-266.1}$	-378.1
DeepSeek V4 Flash	medium	$\mathbf{-247.7}$	-729.5

Detailed analysis shifts returns in different directions. At medium effort, DeepSeek V4 Flash records -247.7 mbb/hand under the short format and -729.5 under EAD, while GPT-5.5 records -271.7 and -155.5, respectively (Table 1b). The expert guidance remains identical across formats, and the changes differ in direction.

Motivation. The same expert-written skill yields different observed returns across models and configurations. With the expert guidance held fixed, these results suggest that a skill's effectiveness depends on how each model applies it. This motivates a way for each model to adapt shared expert guidance to its own play.

3 Recursive Skill Improvement with RIVER

In this section, we propose RIVER, an iterative method in which the LLM that plays poker recursively improves its own skill library without changing its weights. RIVER combines a factorized skill library, ensemble reflection, and hierarchical induction to improve skills across poker's vast decision space under partial observability and high-variance returns. Figure 2 illustrates one round, and Algorithm 1 gives the complete procedure.

3.1 Problem Setup

Let M be a language model and let S be a skill library, a structured natural-language text described in Section 3.2. A fixed poker agent turns each game state x into a textual observation, gives M the observation together with the part of the library relevant to x, written $S(x)$, and parses the response into a legal action. The model and the library together define a policy $\pi_{M,S}$. The goal is a library with a high expected return,

$$J(S) \;=\; \mathbb{E}_{h\sim\pi_{M,S}}\big[\, r(h) \,\big], \tag{1}$$

where h is a hand and $r(h)$ is its return. In round t, the model plays N hands with the library S_t, starting from S_0, and the mean return over these hands is the *rollout return* $\bar{r}_t$. The same model then improves a *base skill* $B_t \in \{S_0, \ldots, S_t\}$ (Section 3.5) into the next version S_{t+1}, using the evidence collected up to round t.

3.2 Factorized Skill Library

A poker decision combines several pieces of information, such as the strength of the hand, the texture of the board, and the betting line. Learning from a decision therefore requires attributing it to the information that drove it, and a skill written as a single block of text offers no structure for this attribution. RIVER factorizes the skill library along the information on which decisions depend. The model can then attribute each reason to one factor, compare decisions along one factor at a time, and confine each revision to one factor skill.

Skill decomposition. The skill library is divided into six *factor skills* S^g, one for each decision factor g (Figure 1a). A decision factor describes one aspect of a decision made after the first public cards are dealt, and its value is computed deterministically from the game state (Appendix B). *Hand* is the made-hand category and drawing potential of the player's cards, such as top pair or a draw. *Board* is the texture of the public cards, such as dry, wet, or paired. *Action* describes the betting line, for instance whether the player faces a bet or is checked to. *Pot & SPR* describes how the pot was

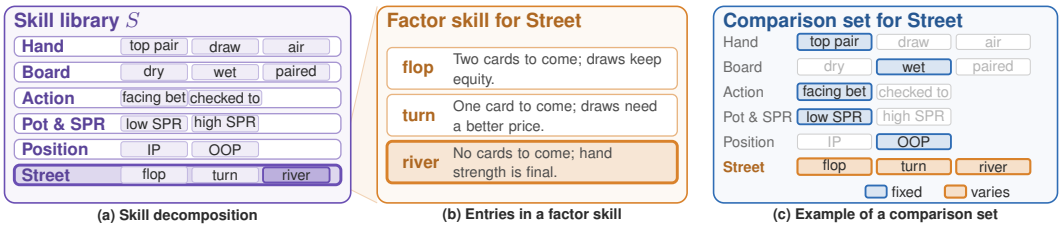

Figure 1: Factorized skill library, with illustrative content. (a) The skill library is divided into six factor skills, one for each decision factor. (b) The Street factor skill holds one entry per street, and a river decision retrieves the river entry. (c) A comparison set with Street as the focus factor fixes the other five factors. Each combination of fixed values defines a different comparison set.

formed and the stack-to-pot ratio (SPR) of the remaining stacks. *Position* records whether the player acts in or out of position and whether it is the aggressor or the defender. *Street* is the flop, the turn, or the river. The six factors describe a decision from complementary angles with little overlap, and together they cover the information on which poker strategy depends.

Factor skills and entries. Each factor skill holds one *entry* for every value of its factor that has occurred in play (Figure 1b), and an entry states how a decision should take this value into account. At each decision, the agent retrieves from every factor skill the entry matching the decision, so a river decision receives the river entry of the Street factor skill, and $S(x)$ combines the six retrieved entries. Each update revises the factor skills one at a time. In the initial version S_0, every factor skill is empty and the library contains only our basic description of the rules of poker, so all postflop strategic content in later versions is generated by RIVER.

Comparison sets. The decomposition lets RIVER attribute reasons to a single factor through comparison. For a *focus factor* g, a *comparison set* fixes one value for each of the other five factors and collects the decisions with these values, so the decisions in the set agree in every factor except g (Figure 1c). Differences among the analyses of these decisions can then be associated with the value of g, and the principles drawn from them go into the factor skill S^g. In Figure 1c, for example, the set collects decisions with top pair on a wet board, facing a bet out of position with a low SPR, on the flop, the turn, and the river. What changes across these decisions then points to the street. Appendix C gives the matching rule and evidence threshold.

3.3 Ensemble Reflection

The return of a poker hand does not verify the decisions made in it, and each available source of evidence judges a decision only in part. Human expertise is general and independent of the current skill, an audit of the model's own play is specific to the current skill yet cannot tell whether the skill itself is sound, and the return judges a whole version yet cannot locate the entries responsible. RIVER therefore reflects on each round through all three as separate evidence streams, and each stream covers blind spots of the other two.

Human-expertise stream. Human expertise enters through the top lane of panel 3 in Figure 2. We use PokerSkill (Li et al., 2026) as the source of expert guidance. Since the guidance leaves open which considerations dominate in a specific situation, we anchor it with *reference decisions*, each a sampled state paired with the action taken in it (Appendix A). For each reference decision, the model receives the PokerSkill guidance available in that state, identifies which parts of the guidance account for the action, and distills a transferable principle that must be grounded in the text of the guidance, so the actions themselves cannot be distilled into the skill. The resulting *expertise cases* bring the reasoning behind expert play into each update of the skill.

Skill-execution stream. The middle lane audits the model's own play. *Execution cases* are the model's decisions under the current library, and they reveal whether the skill was applied as intended, where it was ambiguous, and where it led to poor choices.

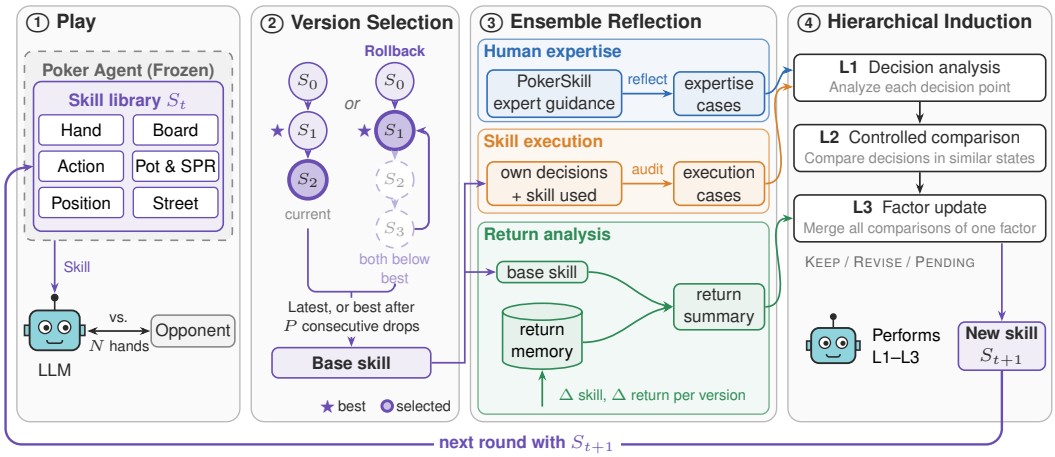

Figure 2: Overview of one RIVER round. (1) The frozen poker agent plays N hands with the skill library S_t. (2) Version selection picks the base skill, a star marking the best version and a bold circle the selected one. (3) Three streams reflect on the round. (4) Hierarchical induction condenses their evidence into the next version S_{t+1}.

Return-analysis stream. Returns are analyzed in the bottom lane, at the level of whole library versions. After the model plays with S_t, we compare the rollout return of S_t with that of its base skill B_{t-1}, and the model writes a short *return analysis* for each factor, stating what changed in its factor skill, how the return moved, and which changes should be preserved, revised, or rechecked. A comparison between two versions is noisy, however, and when several factor skills change at once, their effects mix in the change of return. Separating them would require evaluating each factor on its own, which multiplies the number of evaluations. RIVER therefore appends each return analysis to a *return memory* kept for its factor, which records how past revisions of the factor skill were followed by changes in return and lets the model look for returns that consistently follow them. When a factor skill is updated, the model condenses that factor skill in the base skill, together with the memory of its factor, into a *return summary*.

3.4 Hierarchical Induction

A round produces the records of thousands of hands. Given all of these records at once, the model cannot tell what to attend to when it revises a factor skill. RIVER therefore condenses the evidence level by level through *hierarchical induction* (panel 4 in Figure 2). L1 analyzes each decision on its own, L2 aggregates the analyses of similar decisions within each comparison set, and L3 weighs the resulting summaries of the three streams against one another to revise each factor skill.

Decision analysis (L1). L1 compresses the record of each decision into a short analysis of the reasons behind it. An expertise analysis identifies the reasons for the reference action grounded in the PokerSkill guidance and distills a transferable principle. The reference action is used only in this analysis and is left out of the input to L2. An execution analysis records how the model applied its current skill. Appendix C describes the fields passed between levels, and Appendix F gives the prompt templates.

Comparison-set summary (L2). L2 condenses the analyses in each comparison set (Section 3.2) into principles that recur across hands, separately for the two streams, and the expertise summary keeps each value of the focus factor apart. These principles average out the noise of individual hands, show how the decision changes with the focus factor, and retain the reasons behind the different actions taken in similar states.

Factor update (L3). L3 turns the evidence of the three streams into revised factor skills. For each factor, the model weighs the L2 summaries of its comparison sets against its return summary and marks each entry of the factor skill KEEP, REVISE, or PENDING. The revised factor skills form the next version S_{t+1}.

Algorithm 1 RIVER

Require: model M, fixed poker agent, opponent, number of rounds T, hands per round N, threshold K, patience P

1: $S_0 \leftarrow$ rules description with empty factor skills; expertise and execution pools per comparison set $\leftarrow \emptyset$; return memory per factor $\leftarrow \emptyset$
2: **for** $t = 0, \ldots, T-1$ **do**
3: Play N hands with π_{M,S_t}, record rollout return $\bar{r}_t$, and update best version $S^\star$ ▷ play
4: $B_t \leftarrow S_t$; if the last P versions all fall below $S^\star$, set $B_t \leftarrow S^\star$ ▷ version selection
5: Add reference decisions to the expertise pools ▷ human expertise
6: Add own decisions under S_t to the execution pools, or restore those of $S^\star$ if $B_t \neq S_t$ ▷ skill execution
7: If $t \geq 1$, write a return analysis per factor comparing B_{t-1} and S_t, and append it to the factor's return memory ▷ return analysis
8: **for** each comparison set with at least K cases in both pools **do**
9: Analyze its cases with M ▷ L1
10: Summarize the analyses of each stream with M ▷ L2
11: **end for**
12: **for** each factor g with ready comparison sets **do**
13: Condense B_t^g and the return memory of g into a return summary with M
14: $S_{t+1}^g \leftarrow M(B_t^g,$ L2 summaries of $g,$ return summary$)$, marking each entry KEEP, REVISE, or PENDING ▷ L3
15: If $S_{t+1}^g \neq B_t^g$, discard the execution cases of the comparison sets of g
16: **end for**
17: Factor skills without ready comparison sets are copied unchanged from B_t into S_{t+1}
18: **end for**

3.5 OVERALL PROCEDURE

Version selection. Version selection chooses the base skill from which each update starts (panel 2 in Figure 2), so that poor updates do not build on one another. It tracks the *best version*, the one with the highest rollout return so far. Once P consecutive versions fall below the best version, the next update starts from the best version and restores the execution cases gathered under it, and otherwise it starts from the latest version.

One round. Algorithm 1 and Figure 2 show how the components fit into one round. RIVER starts from the initial library S_0, which contains only the rules description (line 1). In round t, the model plays N hands with S_t and records the rollout return, which also updates the best version (line 3, panel 1), and version selection fixes the base skill B_t (line 4, panel 2). The reference decisions and the model's own decisions are added to the pools of their comparison sets (lines 5–6), and the return-analysis stream appends a return analysis to the memory of each factor (line 7). Each comparison set that has gathered at least K cases in both pools then passes through L1, which analyzes at most K cases of each stream, and L2 (lines 8–10, panels 3 and 4). L3 updates each factor skill in B_t (lines 12–15), factor skills without ready comparison sets are copied unchanged (line 17), and the resulting version S_{t+1} is played in the next round.

4 EXPERIMENTS

We organize the experiments around three questions. Does RIVER improve on the expert-written skill across models and reasoning efforts (RQ1)? How does each component of Section 3 contribute (RQ2)? What does the learned skill library contain (RQ3)?

4.1 EXPERIMENTAL SETUP

Configurations. We run RIVER on four configurations with the short format, namely Gemini 3.7 Flash at low and medium reasoning effort, DeepSeek V4 Flash at low effort, and GPT-6 Luna at medium effort. They cover three model families, and the two Gemini configurations cover both reasoning efforts of the same model.

Table 2: Return (mbb/hand) against Slumbot with the short format. PokerSkill is the expert-written baseline from Table 1, S_0 is the initial library of RIVER, and the last column is the best version of each RIVER run.

Model	Effort	PokerSkill	S_0	RIVER
Gemini 3.7 Flash	low	-54.7	-109.5	$\mathbf{+349.6}$
Gemini 3.7 Flash	medium	-190.4	-215.4	$\mathbf{+232.8}$
DeepSeek V4 Flash	low	-266.1	-287.1	$\mathbf{+260.1}$
GPT-6 Luna	medium	-166.6	-173.5	$\mathbf{+216.7}$

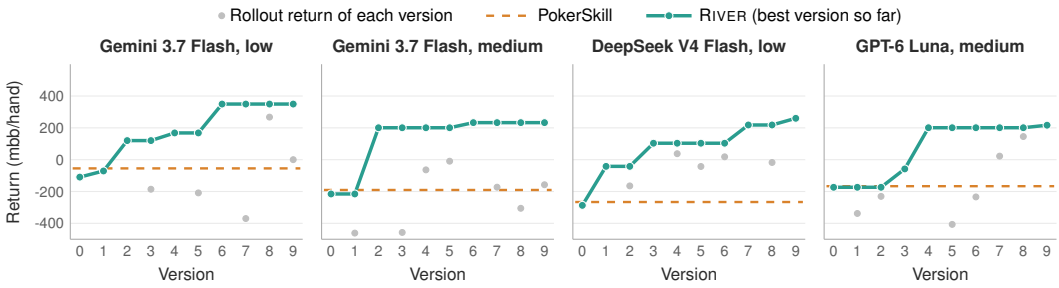

Figure 3: Return of RIVER over versions for each configuration. The solid line is the best version so far, the one retained by version selection, and the gray dots are the rollout returns of individual versions over 5,000 hands each. Version 0 is the initial library S_0, and the dashed line marks PokerSkill.

Evaluation protocol. Each run starts from S_0 and produces nine further versions, with $K = 30$ and $P = 2$. Every version plays 5,000 hands against Slumbot, and we report its mean return in mbb/hand. The baseline for each configuration is PokerSkill with the same model and reasoning effort (Table 1). For each run, we report the return of the best version. RIVER aims to develop a skill that the model can deploy, and since the model rewrites its skill stochastically, the quality of successive versions rises and falls. Version selection retains the version with the highest return, which is the version RIVER would deploy, and Figure 3 also shows the return of every version. Appendix D gives the model request details, and Appendix E discusses the choice of this evaluation.

4.2 Does RIVER Improve on the Expert-Written Skill? (RQ1)

Table 2 and Figure 3 compare PokerSkill with the skills learned by RIVER. In all four configurations, the best version of RIVER reaches a positive mean return and exceeds PokerSkill by 383 to 526 mbb/hand. The initial library S_0, which holds only the rules description, stays close to PokerSkill in every configuration, so the gain comes from the strategic content that RIVER writes. Each model starts from the same S_0 and arrives at a skill of its own that exceeds the expert-written one.

The two Gemini configurations cover both reasoning efforts of the same model. RIVER improves the return at both efforts, from -54.7 to $+349.6$ mbb/hand at low effort and from -190.4 to $+232.8$ at medium effort, so the model reaches a positive mean return at either setting.

The returns in Figure 3 fluctuate from version to version, and a version often falls below the best one, as version 7 does for Gemini 3.7 Flash at low effort. These fluctuations come from both the variance of a 5,000-hand evaluation and the stochastic rewriting of the skill (Appendix E), and version selection keeps each run anchored to its best version.

4.3 How Does Each Component Contribute? (RQ2)

We ablate RIVER on Gemini 3.7 Flash at low effort with three variants. Each variant removes one component of Section 3, keeps the rest of the method unchanged, and runs for ten versions under the same protocol (Figure 4). The best version of RIVER reaches $+349.6$ mbb/hand.

w/o expertise removes the human-expertise stream and keeps the skill-execution and return-analysis streams, so each update draws only on the model's own decisions and on returns. Since S_0 holds no strategy, this variant tests whether the model can build a skill without principles from expert guidance. Its best version reaches $+142.5$. Without expertise cases, the model has to derive every principle from its own play, which suggests that expert evidence supplies content the model does not derive on its own.

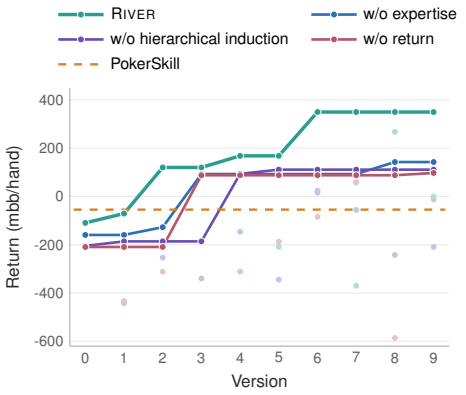

Figure 4: Ablations on Gemini 3.7 Flash at low effort. Lines show the best version so far, and dots show the return of each version. The w/o return variant has no version selection.

w/o hierarchical induction keeps the factorized library, the human-expertise stream, and the return analysis, and replaces the condensed evidence of L1 and L2 with randomly sampled trajectories of the model's own play, which the model reads directly when it updates each factor skill. This variant tests whether the evidence needs to be condensed before an update. Its best version reaches $+110.9$. Raw trajectories leave the model to find the recurring reasons across decisions by itself, without comparison sets to organize them, which suggests that condensed evidence gives the model a clearer basis for revising each factor skill.

w/o return removes the return-analysis stream with version selection and keeps the other two streams, so no version-level return enters the update and each update starts from the latest version. This variant tests whether returns are needed to check updates. Its best version reaches $+97.2$. Without returns, the model cannot tell which revisions were followed by a drop and may keep building on poor versions, which version selection in RIVER prevents by returning to the best version.

Given the high variance of poker and the stochasticity of the model, we read these results as a consistent direction, in which removing any of the three components lowers the best return.

4.4 WHAT DOES RIVER LEARN? (RQ3)

We examine the skill library that Gemini 3.7 Flash learns at low effort. PokerSkill states general principles that leave the choice of action open in many situations, and the learned entries settle these choices for specific factor values.

Stabbing in limped pots (Pot & SPR). PokerSkill gives a single rule for hands without a made hand when the opponent checks, to stab "at HIGH frequency." The learned entry keeps stabs for boards that strongly favor the player's range and has unmade trash "prioritize taking a free card by checking behind." In a limped pot neither player has shown strength before the flop, so a check does not mark the opponent as weak, and a stab with trash on a board that does not favor the player is often called or raised. The entry ties the stab to the board, a condition PokerSkill leaves unstated.

Medium-strength hands (Action). PokerSkill states that such hands "mostly aim for cheap showdown" but leaves open how to play them when checked to. The learned entry specifies checking behind with second pair, top pair with a weak kicker, and third pair, which controls the pot and avoids check-raises. These hands win at showdown often enough to prefer a cheap one, while a bet is mostly called by better hands. The entry turns the general aim of PokerSkill into a concrete action for the spot where the opponent checks.

Bluffing on the river (Board). PokerSkill allows betting most hands without showdown value and gives blocker-based advice for choosing bluffs as a separate rule. The learned entry combines the two for boards with three of a kind and three cards of one suit, where a hand without a card of that suit blocks little value, and checks with Ace-high and pure trash. On such boards the opponent's strongest hands are full houses and flushes, so a bluff works best when it holds a card the opponent needs for them. The entry selects bluffs by their blockers and gives up with hands that block nothing.

In all three cases RIVER adds the condition under which the expert advice applies. Such open choices leave room for different models to act differently under the same skill, consistent with Section 2, and RIVER lets each model settle them for itself.

5 Related Work

AI for poker. Superhuman poker agents such as DeepStack (Moravčík et al., 2017), Libratus (Brown & Sandholm, 2018), and Pluribus (Brown & Sandholm, 2019) rely on equilibrium approximation and search, and their evaluations use variance-reduction techniques such as AIVAT (Burch et al., 2018) because raw winnings are noisy. Recent work brings LLMs to the game. PokerBench (Zhuang et al., 2025) fine-tunes LLMs on solver-labelled decisions, LeAct (Yang et al., 2026b) trains them on reasoning recovered from solver actions, and ToolPoker (Lin et al., 2026) calls solvers at decision time. Suspicion-Agent (Guo et al., 2023) prompts GPT-4 to model the opponent's beliefs, and Agent-Pro (Zhang et al., 2024) refines its behavioral guidelines through reflection over past games of Limit hold'em. PokerSkill (Li et al., 2026) lets frontier LLMs play competitively with an expert-written skill, and our work starts from its released agent and skill. RIVER keeps the model weights fixed and treats the noisy return only as version-level evidence. Different from Agent-Pro, which reflects only on its own games of four-player Limit hold'em, and LeAct, which reasons from expert actions to train the model weights, RIVER iteratively improves a factorized natural-language skill from three evidence streams and scales to full heads-up no-limit hold'em.

Prompt, context, and harness optimization. LLM-driven optimizers search over text or code using a score for each candidate. OPRO (Yang et al., 2024) conditions on past prompts and their scores, TextGrad (Yuksekgonul et al., 2024) propagates textual feedback through a computation graph, Promptbreeder (Fernando et al., 2023) co-evolves task and mutation prompts, and GEPA (Agrawal et al., 2026) reflects on execution traces and keeps a Pareto front of candidates. ACE (Zhang et al., 2026b) evolves a context playbook through incremental edits, and Meta-Harness (Lee et al., 2026) uses a coding agent to search over the harness of a target model. These methods assume a reliable score for each candidate. In poker such a score is unavailable for single decisions and noisy for whole versions, and RIVER compensates with evidence from three complementary streams.

Self-improving agents and skills. Agent skills package procedural knowledge as natural-language instructions that an agent loads at inference time (Xu & Yan, 2026). SkillGrad (Wang et al., 2026) and SkillOpt (Yang et al., 2026a) optimize the skill of a frozen agent, SkillGrad from losses on tasks with ground-truth answers and SkillOpt by accepting an edit only when it improves a held-out validation score. Reflexion (Shinn et al., 2023) turns feedback into verbal self-reflection, ExpeL (Zhao et al., 2024) extracts reusable insights from past trajectories, and Voyager (Wang et al., 2023) accumulates a library of executable skills. Self-Harness (Zhang et al., 2026a) lets a fixed model refine its own harness, and STOP (Zelikman et al., 2023) applies a self-improving scaffold to itself. RIVER is a form of recursive self-improvement in which the model that plays rewrites its own skill library, organized by decision factor so that each change stays within one factor skill.

6 Conclusion

We find that the return of an expert-written poker skill varies across base models, across reasoning efforts of a single model, and across requests for more detailed analysis before acting, so the value of a skill depends on how each model executes it. We introduce RIVER, in which the LLM that plays poker recursively improves its own skill library without changing its weights. Through a factorized skill library, ensemble reflection, and hierarchical induction, RIVER learns from noisy, indirect evidence and organizes the skill to cover a vast decision space. Across four configurations spanning three model families, RIVER derives for every model a skill that reaches a positive mean return against the Slumbot bot and exceeds PokerSkill by 383 to 526 mbb/hand. These results suggest that a model can start from human expertise and iteratively derive a poker skill suited to itself.

7 Limitations.

We study heads-up no-limit hold'em, and extending RIVER to multi-player poker remains open.

AI USE STATEMENT

Generative AI tools assisted with mathematical proof checking, code verification, and manuscript polish. They were not used to generate experimental data or results. The authors reviewed and verified all AI-assisted outputs and take full responsibility for the final content of the paper.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.

Neil Burch, Martin Schmid, Matej Moravcik, Dustin Morill, and Michael Bowling. Aivat: A new variance reduction technique for agent evaluation in imperfect information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Paul Glasserman and David D Yao. Some guidelines and guarantees for common random numbers. *Management Science*, 38(6):884–908, 1992.

Jiaxian Guo, Bo Yang, Paul Yoo, Bill Yuchen Lin, Yusuke Iwasawa, and Yutaka Matsuo. Suspicion-agent: Playing imperfect information games with theory of mind aware gpt-4. *arXiv preprint arXiv:2309.17277*, 2023.

Eric Jackson. Slumbot nl: Solving large games with counterfactual regret minimization using sampling and distributed processing. In *AAAI Workshop on Computer Poker and Incomplete Information*, 2013.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Boning Li, Baoxiang Wang, and Longbo Huang. Pokerskill: Llms can play expert-level poker without training or solvers. *arXiv preprint arXiv:2605.30094*, 2026.

Minhua Lin, Enyan Dai, Hui Liu, Xianfeng Tang, Yuliang Yan, Zhenwei Dai, Jingying Zeng, Zhiwei Zhang, Fali Wang, Hongcheng Gao, et al. How far are llms from professional poker players? revisiting game-theoretic reasoning with agentic tool use. *arXiv preprint arXiv:2602.00528*, 2026.

Michael Littman and Martin Zinkevich. The 2006 aaai computer-poker competition. *ICGA Journal*, 29(3):166–167, 2006.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. Skillgrad: Optimizing agent skills like gradient descent. *arXiv preprint arXiv:2605.27760*, 2026.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 12028–12068, 2024.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026a.

Ziran Yang, Chengshuai Shi, Raj Ghugare, Benjamin Eysenbach, Karthik Narasimhan, and Chi Jin. Leact: Learning to reason from expert actions. *arXiv preprint arXiv:2607.21856*, 2026b.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026b.

Wenqi Zhang, Ke Tang, Hai Wu, Mengna Wang, Yongliang Shen, Guiyang Hou, Zeqi Tan, Peng Li, Yueting Zhuang, and Weiming Lu. Agent-pro: Learning to evolve via policy-level reflection and optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5348–5375, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Richard Zhuang, Akshat Gupta, Richard Yang, Aniket Rahane, Zhengyu Li, and Gopala Anumanchipalli. Pokerbench: Training large language models to become professional poker players. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26175–26182, 2025.

A Reference Decisions

In our experiments, reference decisions come from the opponent, Slumbot (Jackson, 2013), whose hole cards are revealed when a hand ends. For each of its decisions, including folds, we reconstruct the state it faced from its own perspective, with the public cards truncated to the current street so that the reconstruction contains no information from later streets. We then generate the PokerSkill guidance available in that state and pair it with the action Slumbot took, which forms the reference action.

The reconstructed state contains only the information available to Slumbot when it acted, namely its own hole cards, the public cards up to the current street, and the preceding actions. L1 receives this state and the reference action as separate inputs (Section 3.4) and uses the PokerSkill guidance to write a candidate explanation of the reference action and a transferable principle grounded in that guidance. The reference action is left out of the input to L2 and L3, which receive the resulting case analyses and summaries and are instructed not to turn actions into action rules (Appendix F).

B SCOPE AND FACTORS OF THE SKILL LIBRARY

The factorized skill library of Section 3.2 covers the decisions made after the first public cards are dealt, that is, on the flop, the turn, and the river. Decisions before the flop follow the preflop guidance of the PokerSkill prompt (Li et al., 2026), which lies outside the skill library and stays the same in every version and in every configuration, including the PokerSkill baseline. Differences in strategy between RIVER and PokerSkill therefore lie in decisions after the first public cards are dealt.

Preflop play in no-limit hold'em is largely standardized. A preflop decision is determined by one of a small number of fixed scenarios, such as opening the pot or facing a raise from a given position, together with one of 169 strategically distinct starting hands. This space is small and good play within it follows well-established rules, so RIVER reuses the preflop guidance of the PokerSkill prompt. The difficulty that motivates RIVER lies after the flop, where the public cards, the betting line, and the stack depth combine into a far larger space of situations. Before the flop, moreover, no public cards have been dealt, so the Board factor carries no information and the Street factor takes a single value.

The six factors are defined in more detail as follows. *Hand* is the made-hand category and drawing potential of the player's cards. *Board* is the texture of the public cards, namely their suit distribution, pairing, and connectivity. *Action* describes the betting line, including whether the player faces a bet and whether the situation calls for a proactive, defensive, or counter-aggressive decision. *Pot & SPR* describes how the pot was formed and the stack-to-pot ratio of the remaining stacks. *Position* records whether the player acts in or out of position and whether it is the aggressor, the defender, or neither. *Street* is the flop, the turn, or the river.

Each factor value is a tuple of the fields below, and a factor-skill entry is keyed by that tuple. The Hand, pot-type, and role labels are extracted from the state text produced by the PokerSkill context engine. The remaining fields use the rules described here.

Hand. The made-hand field reads the first component of the `Hand evaluation` line, with a draw-only first component mapped to `NO MADE HAND`. A separate draw field reads the first draw component of that line, or `NONE` if no draw is listed. The context engine supplies labels such as `TOP PAIR`, `SET`, and `STRONG DRAW`; RIVER uses these labels as its hand categories.

Board. The suit field records the largest number of public cards sharing a suit. The pairing field sorts the multiplicities of public-card ranks, giving patterns such as `1-1-1` on an unpaired flop and `2-2-1` on a double-paired river. The connectivity field counts the largest number of distinct public-card ranks in any five-rank straight window. An ace is considered both high and low for this calculation. These three fields retain separate information about suit concentration, pairing, and straight potential.

Action. The facing-bet field records whether the current decision prompt contains `Facing bet of`. The decision regime is `proactive` when the player faces no bet, `counter_aggressive` when the player faces a bet and raising is legal, and `defensive` when the player faces a bet and raising is unavailable.

Pot and SPR. The pot-type field reads the context-engine label from `LIMP POT`, `SRP`, `3BP`, `4BP`, and `5BP PLUS`. The SPR field places the printed stack-to-pot ratio into four intervals, $[0, 1)$, $[1, 3)$, $[3, 6)$, and $[6, \infty)$.

Position. The position field maps `BTN` to in position and all other position labels to out of position. The role field reads the context-engine label `AGGRESSOR`, `DEFENDER`, or `NEUTRAL`.

Street. The street field takes one of `flop`, `turn`, and `river`. Only decisions on these streets enter the factorized library and induction cases.

C Comparison Sets and Skill Induction

For a focus factor g, a comparison set fixes the values of the other five factors and collects cases with those values across all observed values of g. Each postflop decision contributes to one set for each choice of focus factor. The set compares decisions at the resolution of the factor categories above, so cases can still differ in cards, bet sizes, and other details within a category.

Let $E(C)$ be the number of expertise cases accumulated for set C and $X_t(C)$ the number of execution cases available under the current skill evidence pool. The set is eligible for induction in round t when $E(C) \geq K$ and $X_t(C) \geq K$, with $K = 30$ in our experiments. Expertise cases accumulate across rounds. Execution evidence tracks the current skill. It is discarded when guidance in its factor skill changes and restored to that of the best version when version selection returns to it. For each eligible set, L1 analyzes at most 30 expertise cases and 30 execution cases.

The expertise L1 request supplies the decision-time observation, the PokerSkill guidance for that state, and the reference action as separate inputs. Its `observation` output refers to the state available before the decision. Its `candidate_reasoning` is instructed to explain which PokerSkill considerations could support the action; `generalizable_principle` expresses a transferable condition or trade-off grounded in that guidance. The output also records whether the case supports such a principle, conflicts with it, or offers insufficient evidence. The execution L1 request examines the target model's action and reasoning together with the matched current skill and produces an execution assessment and a revision signal. At L2, each stream condenses its case analyses within the same comparison set. The expertise summary separates focus-factor combinations and their evidence counts; the execution summary describes how the current skill was applied. L3 receives these summaries and the factor's return summary to update that factor skill.

The initial library has empty factor-skill sections. When an observed factor value first appears in L2 evidence, the corresponding entry can be created with status PENDING. A Pending entry retains its key while contributing no entry-specific guidance to the playing prompt; any group-wide guidance still applies. The generator reviews each entry separately and assigns KEEP, REVISE, or PENDING. Its guidance text determines the operative skill. A change to that text advances the factor-skill version and discards its execution evidence; the status label records the generator's assessment. The return memory of each factor persists across versions and provides version-level context for later updates.

D Model Requests and Evaluation Handling

The four RIVER runs each begin with their own S_0 rollout and generate versions S_1 through S_9 sequentially. Each version is evaluated over 5,000 completed hands against Slumbot. Table 3 gives the model identifier and reasoning effort sent for both play and induction. The requests use an OpenAI-compatible Responses endpoint with `model`, `input`, `max_output_tokens`, `reasoning.effort`, `temperature`, and `top_p`. Play requests allow up to 4,096 output tokens and induction requests up to 32,768. Sampling uses a temperature of 1 and a top-p of 1.

Table 3: Requested model identifiers and reasoning efforts in the four RIVER runs.

Model in the paper	Requested model identifier	Reasoning effort
Gemini 3.7 Flash	`gemini-3.7-flash`	low
Gemini 3.7 Flash	`gemini-3.7-flash`	medium
DeepSeek V4 Flash	`deepseek-v4-flash`	low
GPT-6 Luna	`paid/gpt-6-luna`	medium

E Variance of Evaluation

Returns in heads-up no-limit hold'em have high variance. The return of a hand depends heavily on the private and public cards, and a single hand can move up to the full stack of 200 big blinds. An LLM player adds a further source of variation, since the same model may choose different actions

in the same state. Following PokerSkill (Li et al., 2026), we evaluate each skill version over 5,000 hands.

Standard variance-reduction techniques for poker do not apply to this evaluation. AIVAT (Burch et al., 2018) requires the action probabilities of the evaluated agent at every decision and a value estimate for game states, and an LLM player provides neither. Duplicate poker (Littman & Zinkevich, 2006) plays each deal twice with the seats swapped, and common random numbers (Glasserman & Yao, 1992) evaluate the compared agents on the same deals. Both require control over the cards, whereas the Slumbot API deals every hand on its server and accepts no seed. We therefore report the raw mean return, as PokerSkill does.

The output of RIVER is a skill for the model to deploy. Each rewrite of the skill by the model is stochastic, so the versions produced by the same procedure differ in quality, and a developer deploys the best version found by the end of the iteration. We select this version by its rollout return, the same version that version selection retains during the iteration, and we report its return.

This variance shapes the design of RIVER. The return enters each update only as version-level evidence accumulated in the return memory (Section 3.3), and version selection keeps the iteration from building on versions whose returns keep falling (Section 3.5).

F Prompt Templates

The following unfilled templates supply the system and user messages for L1, L2, return analysis, and L3. Braced fields are populated with the current case, skill, or summaries at request time. The templates are shown before substitution.

F.1 L1 Expertise Case Analysis

System message.

```
You are analyzing one poker decision as a strategy-distillation assistant.
The Slumbot action is an observed expert behavior, not a guaranteed correct label and not a
     command to imitate. PokerSkill is a human-designed strategic reference, not proof that
     every individual action is optimal.
Explain which PokerSkill guidance could plausibly support the observed behavior. Distill
     decision principles, conditions, and trade-offs—not a fixed action or bet size.
Return JSON only with exactly these fields:
{
  "observation": string,
  "candidate_reasoning": string,
  "generalizable_principle": string,
  "support_level": "supported" | "conflicting" | "insufficient_evidence"
}
Do not claim to know Slumbot's internal reasoning. Do not use hidden cards, future cards,
     final outcome, or information unavailable at the decision point.
```

User message.

```
Analyze this one postflop case.

OBSERVATION (visible state, already converted to text):
{observation}

POKERSKILL GUIDANCE AVAILABLE FOR THIS STATE:
{teacher_guidance}

OBSERVED SLUMBOT ACTION (evidence only; do not treat it as a mandatory target):
{expert_action}

If the observed action is a bet or raise, its amount is the total bet/raise-to amount on the
      current street in BB, not the additional increment. A non-aggressive action has no
     sizing amount.

Write a concise candidate explanation. The generalizable_principle must describe what to
     evaluate, how the factors interact, and when the principle applies. Do not write
     instructions such as 'always check' or 'always bet'. If the evidence does not support a
     transferable principle, use insufficient_evidence.
```

F.2 L1 Skill Execution Analysis

System message.

```
Analyze how a poker agent applied its current strategy Skill in one decision.
Use the observation, action, reasoning, and matched Skill to explain the execution and
     identify a useful revision signal for the next Skill update.
Return JSON only with exactly these fields:
{
  "target_reasoning": string,
  "skill_used": string,
  "execution_analysis": string,
  "execution_status": "followed" | "partially_followed" | "not_followed" | "unclear",
  "revision_signal": string
}
```

User message.

```
Analyze this target-model decision.

OBSERVATION:
{observation}

ACTION AND AMOUNT:
{action}

TARGET REASONING:
{target_reasoning}

MATCHED CURRENT SKILL:
{skill_used}

Explain whether the action and reasoning applied the matched Skill appropriately, and
     describe any concrete issue that should be considered in a future Skill update. If the
     evidence does not indicate a problem, say so.
```

F.3 L2 Expertise Comparison-Set Summary

System message.

```
You summarize supported, case-level strategy evidence for one fixed comparison slice.
The cases were analyzed separately. Do not reconstruct or guess the expert's actions. Do not
      turn action mentions or frequencies into action rules.
Extract only recurring decision principles, conditions, trade-offs, and boundaries that are
     supported by the supplied cases. If evidence is weak or inconsistent, say so and do not
     invent a rule.
Return JSON only with exactly these fields:
{
  "slice": string,
  "combination_summaries": {string: {
    "case_count": integer,
    "supported_case_count": integer,
    "conflicting_case_count": integer,
    "insufficient_case_count": integer,
    "principles": [string],
    "conditions": [string]
  }},
  "cross_combination_observations": [string],
  "scope": string
}
```

User message.

```
Summarize the supported evidence for this comparison slice.

FOCUS GROUP: {focus_group}
FIXED OTHER-GROUP CONTEXT: {slice_context}
FOCUS VALUES PRESENT IN THIS SLICE:
{focus_values_present}
FOCUS COMBINATIONS PRESENT IN THIS SLICE:
{focus_combinations}
RAW CASE COUNT: {raw_case_count}

CASE ANALYSES (each includes its joint combination and support status):
{case_json}
```

```
For every focus combination listed in the cases, write a separate entry in
combination_summaries. Do not merge principles from different combinations.
Use the support status and counts as evidence quality information; do not
invent an evidence-strength label. Do not propose a final Skill rewrite and do
not prescribe a fixed action.
```

F.4 L2 Skill Execution Summary

System message.

```
Summarize target-model Skill execution for one fixed comparison context.
The cases share the same non-focus factor values; the focus factor may vary.
Use the supplied execution analyses to describe how the current Skill was applied and
     whether the evidence supports a revision.
Return JSON only with exactly these fields:
{
  "slice": string,
  "execution_summary": string,
  "revision_summary": string,
  "scope": string
}
```

User message.

```
Summarize these target execution analyses.

FOCUS GROUP: {focus_group}
FIXED NON-FOCUS CONTEXT: {slice_context}
FOCUS VALUES PRESENT: {focus_values_present}
CASE COUNT: {raw_case_count}

TARGET ANALYSES:
{case_json}

State what the cases collectively show about applying the current Skill and whether they
     provide a concrete basis for revising it.
```

F.5 Direct Return Analysis

System message.

```
Analyze the version-level return change for one factor Skill.
The previous Skill is the Skill used for the earlier rollout, and the current Skill is the
     Skill used for the most recent rollout. Compare these two Skill versions and their
     observed rollout returns. Identify the concrete guidance changes between them and
     explain which of those changes may be worth preserving, revising, or reverting in the
     next Skill update.
If a change in the current Skill appears harmful, describe reverting that specific change to
      the corresponding guidance from the previous Skill. Do not discuss or assume any
     content that has not yet been generated in the next Skill update.
Use MBB per hand as the primary return signal. Do not consider or discuss confidence
     intervals; poker variance makes them unhelpful for this comparison.
Return JSON only with exactly these fields:
{
  "skill_difference": string,
  "return_change": string,
  "revision_advice": string
}
```

User message.

```
Analyze this factor Skill version comparison.

FOCUS GROUP: {focus_group}
PREVIOUS SKILL:
{previous_skill}

CURRENT SKILL USED FOR THE LATEST ROLLOUT:
{current_skill}

RETURN COMPARISON:
{return_comparison}
```

```
ATTRIBUTION NOTE:
{attribution_note}

Describe the meaningful guidance differences between the previous Skill and the current
     Skill, the observed MBB/hand change, and which current-version changes should be
     preserved, revised, or reverted to the previous-version guidance in the next update. Do
     not prescribe changes that refer to an as-yet-uncreated Skill version. Use MBB per hand
     as the primary return signal. Do not consider or discuss confidence intervals or
     confidence-interval overlap. Treat the return as a version-level signal and use the
     attribution note when interpreting the change.
```

F.6 MEMORY-BASED RETURN ANALYSIS

System message.

```
Synthesize historical version-level return analyses for one factor Skill.
The history contains analyses produced for earlier adjacent Skill transitions in this same
     factor group. Treat them as advisory, version-level evidence rather than proof of
     causality. Identify repeated improvements, repeated failures, and directions that should
      be reconsidered for the current Skill. Use the current Skill as the object to revise;
     do not invent a future Skill version and do not prescribe a change outside the current
     factor group.
Use MBB per hand as the primary return signal. Do not consider or discuss confidence
     intervals; poker variance makes them unhelpful for this comparison.
Return JSON only with exactly these fields:
{
  "skill_difference": string,
  "return_change": string,
  "revision_advice": string
}
```

User message.

```
Generate a memory-based return analysis for this factor Skill.

FOCUS GROUP: {focus_group}
CURRENT ACTIVE SKILL:
{current_skill}

HISTORICAL RETURN ANALYSIS MEMORY FOR THIS FOCUS GROUP:
{history}

Use the historical analyses to identify patterns across prior transitions. Distinguish
     directions that were repeatedly helpful from directions that repeatedly harmed returns
     or were unstable. Then give concrete, narrow advice for updating the current Skill. If
     the history is empty, state that there is no historical return evidence and do not
     invent one.
```

F.7 L3 FACTOR-SKILL GENERATOR

System message.

```
You maintain one factorized poker strategy Skill group.
The supplied slice summaries are distilled evidence from human-expert-guided analyses of
     observed expert behavior. They are hypotheses about decision principles, not action
     labels and not proof of optimal play.
Use them to improve the Skill's decision guidance, not to imitate any observed agent's
     actions. Do not use action frequencies as targets. Do not create new factor groups, new
     trigger fields, or new category values.
The output must remain self-contained: the public guidance and every revised section must
     state its own conditions. Do not rely on external expert text at runtime.
If evidence is insufficient, keep the current Skill unchanged. Prefer a narrow conditional
     principle over a broad rule. Preserve valid alternatives and mixed strategies.

The current group has multiple existing subsections. Generate the complete group
in one call, but audit every subsection independently before deciding the group
result. A summary is direct evidence only for the subsection values listed in
its combination_summaries field. Do not transfer evidence from one subsection to
another merely because both belong to this group. Subsections may overlap when
the same decision genuinely satisfies both conditions. Guidance may refer to
other factor groups because poker decisions combine factors; those references
are conditions, not new triggers.
Return JSON only with exactly these fields:
```

```
{
  "decision": "KEEP" | "REVISE",
  "global_guidance": string,
  "sections": {string: {"status": "KEEP" | "REVISE" | "PENDING", "guidance": string | null
     }},
  "change_rationale": string,
  "evidence_used": [string]
}
Only use section names already present in CURRENT SKILL. A PENDING section must have null
     guidance.
```

User message.

```
Update the focus factor group using the summarized evidence.

FOCUS GROUP: {focus_group}
FOCUS GROUP DEFINITION:
{group_definition}

EXISTING SUBSECTION DEFINITIONS:
{subsection_definitions}

For each subsection, first identify the summaries whose
combination_summaries actually include that subsection value. Then assess the
evidence for that subsection separately. Keep or mark PENDING when there is no
adequate direct evidence. Do not fill every subsection just because another
subsection has strong evidence.

CURRENT SKILL:
{current_skill}

THIS ROUND'S SLICE SUMMARIES:
{slice_summaries}

TARGET EXECUTION SUMMARIES:
{target_summaries}

MEMORY-BASED RETURN ANALYSIS:
{return_analysis}

Use the teacher summaries to identify decision principles, the target summaries to assess
     how the current Skill was applied, and the memory-based return analysis as a historical
     version-level performance signal when deciding the next Skill update. Keep the guidance
     grounded in the supplied evidence.
```