PRODUCT REQUIREMENTS DOCUMENT

Hierarchical Multi-Agent Codeforces Solver

with Session-Scoped Memory and Verification Layer

Version 1.0 | February 2026 | CONFIDENTIAL

Document Type	Product Requirements Document
Version	1.0
Status	Draft — For Review
Author	Product Team
Date	February 2026
Classification	Confidential

1. Executive Summary

This document defines the product requirements for the Hierarchical Multi-Agent Codeforces Solver — a distributed AI system designed to solve competitive programming problems from the Codeforces platform in real time. The system combines a corporate-style agent hierarchy, session-isolated structured memory, execution-based verification, and ELO-inspired performance tracking to deliver accurate, cost-efficient, and explainable solutions.

The core insight is that competitive programming and strategic reasoning share structural parallels: both involve pattern recognition, adversarial evaluation, resource constraint management, and multi-branch exploration. By mapping algorithmic problem-solving onto a hierarchical organization model — and optionally onto chess strategic archetypes — the system achieves greater reasoning robustness and reduced hallucination compared to monolithic LLM approaches.

1.1 Problem Statement

Large language models applied monolithically to competitive programming problems suffer from:

- Hallucinated solutions that pass superficial checks but fail on edge cases
- No structured mechanism for escalating difficult sub-problems
- Unbounded token costs with no adaptive control
- Lack of execution-level validation — logical plausibility is not the same as correctness
- No persistent learning or rating improvement across sessions

1.2 Proposed Solution

A hierarchical multi-agent system where: specialized agents process sub-tasks at different complexity tiers; an orchestrator activates only the agents required for the given problem difficulty; an isolated execution sandbox validates all generated code; and an ELO-style rating manager tracks performance over time.

2. Goals and Success Metrics

2.1 Primary Goals

- Achieve high solve rates across Codeforces problem ratings from 800 to 2400+
- Minimize token expenditure through adaptive agent activation
- Guarantee solution correctness through mandatory execution verification
- Maintain bounded computation cost per problem session
- Provide meaningful, chess-inspired explanations for pedagogical use

2.2 Success Metrics

Metric	Definition	Target
Solve Rate (≤1200)	% of problems solved in ≤2 attempts	≥ 90%
Solve Rate (1200–1800)	% of problems solved	≥ 75%
Solve Rate (1800–2200)	% of problems solved	≥ 55%
Solve Rate (>2200)	% of problems solved	≥ 30%
Avg Token Usage	Mean tokens per solved problem	< 8,000
Escalation Rate	% of sessions triggering H4+	< 20%
Verification Fail Rate	% of accepted solutions that fail judge	< 2%
Avg Cost per Problem	USD at current model pricing	< $0.05

2.3 Non-Goals

- This system is not designed to replace human competitive programmers
- It does not support interactive contest participation with real-time submission
- It does not handle output-only or interactive problem types in the initial release
- The chess abstraction layer does not influence correctness — it is explanatory only

3. Stakeholders

Stakeholder	Role	Primary Interest
Product Team	Owner	Feature completeness, release timeline
ML Engineering	Builder	Model integration, agent design, token efficiency
Platform Engineering	Builder	Sandbox security, scalability, API reliability
QA / Evaluation	Validator	Correctness benchmarking, regression testing
End Users	Consumer	Accurate solutions, clear explanations
Educators / Tutors	Secondary	Chess abstraction layer, pedagogical clarity

4. System Architecture Overview

The system comprises six core components operating as loosely coupled microservices, coordinated by a central orchestrator:

Component	Responsibility
API Gateway	Receives solve requests, fetches problem data from Codeforces, returns structured responses
Orchestrator Service	Initializes sessions, activates agent tiers by rating, enforces budgets, manages escalation
Session Memory Manager	Maintains isolated per-problem workspace with structured JSON files
Agent Pool	Role-specialized agents (H1–H5) that transform session memory incrementally
Execution Sandbox	Compiles and tests generated C++ code in a secure, resource-limited container
Verification Engine	Validates solutions via sample tests, randomized stress tests, and adversarial edge cases
Rating Manager	Maintains ELO-style ratings per agent tier; updates after each session
Token Budget Monitor	Enforces per-role token caps; triggers summarization; tracks cost per session

5. Component Requirements

5.1 API Gateway

Functional Requirements

- FR-GW-01: Accept a solve request containing problem_id and optional contest_id
- FR-GW-02: Fetch problem statement, constraints, sample I/O, and tags from Codeforces API (problemset.problems, contest.list, contest.standings)
- FR-GW-03: Return a structured response containing: final solution code, natural-language explanation, confidence score (0.0–1.0), and rating delta
- FR-GW-04: Expose health-check and metrics endpoints

Non-Functional Requirements

- NFR-GW-01: API response time for problem fetch ≤ 2 seconds (p95)
- NFR-GW-02: Support concurrent requests ≥ 100 via async queue
- NFR-GW-03: Retry Codeforces API on transient failures with exponential backoff

5.2 Orchestrator Service

Functional Requirements

- FR-OR-01: Initialize an isolated session workspace for each incoming problem
- FR-OR-02: Determine the active agent set $\alpha(R_p)$ based on problem rating:
  - $R_p < 1200 \rightarrow$ activate {H1, H2}
  - $1200 \leq R_p < 1800 \rightarrow$ activate {H1, H2, H3}
  - $1800 \leq R_p < 2200 \rightarrow$ activate {H1, H2, H3, H4}
  - $R_p \geq 2200 \rightarrow$ activate full stack {H1, H2, H3, H4, H5}
- FR-OR-03: Route session memory files to each agent in hierarchy order
- FR-OR-04: Trigger escalation when confidence $< \theta$ or verification fails
- FR-OR-05: Enforce maximum 2 revision cycles before hard termination
- FR-OR-06: Call Rating Manager to update ELO after session completes
- FR-OR-07: Reset and archive session workspace after finalization

Escalation Rules

Escalation is permitted only upward in the hierarchy. The depth cap is $d_{max} = 2$. If $\phi(\Sigma_{i+1}) \geq \phi(\Sigma_i)$ is not satisfied after one escalation cycle, the session is marked UNRESOLVED and terminated.

5.3 Session Memory Manager

Workspace Structure

Each session uses the following file structure:

- /session_id/problem_spec.json — raw problem data from API
- /session_id/constraint_analysis.json — extracted constraints (H1 output)
- /session_id/candidate_strategies.json — proposed + refined strategies

- /session_id/decision_log.json — append-only log with reason fields
- /session_id/proof_notes.md — correctness proofs (H3 output)
- /session_id/test_results.json — sandbox execution results
- /session_id/final_solution.cpp — accepted C++ solution

Functional Requirements

- FR-SM-01: All files must be structured JSON or Markdown — no free-form text
- FR-SM-02: decision_log.json must be append-only; each entry must include a reason field
- FR-SM-03: No cross-session file access is permitted
- FR-SM-04: Sessions may be exported for audit; they are discarded by default after completion
- FR-SM-05: Version-controlled edits required — each write must include agent ID and timestamp

5.4 Agent Pool

The agent hierarchy is ordered H1 ≺ H2 ≺ H3 ≺ H4 ≺ H5. Finalization authority flows strictly upward. Each agent receives only the session files relevant to its role.

Agent	Tier	Responsibilities	Output Schema
Intern	H1	Extract constraints, propose paradigms, estimate complexity	proposed_paradigm, estimated_complexity, confidence
Engineer	H2	Refine strategies, eliminate infeasible, structure algorithm design	refined_strategy, complexity_class, eliminated_options
Senior	H3	Prove correctness, identify edge cases, optimize complexity	proof_sketch, edge_cases, optimized_complexity
Lead	H4	Compare multiple strategies, select optimal, validate scalability	selected_strategy, strategy_ranking, trade_off_rationale
CEO	H5	Final approval, confidence calibration, trigger rating update, generate explanation	final_solution, confidence, explanation, rating_delta

5.5 Execution Sandbox

Functional Requirements

- FR-SB-01: Compile C++ code using g++ with standard competitive programming flags
- FR-SB-02: Execute compiled binary against all sample test cases
- FR-SB-03: Generate at least 100 randomized test cases within problem constraints
- FR-SB-04: Synthesize adversarial edge cases (boundary values, empty input, maximum N)
- FR-SB-05: Report PASS, FAIL, TLE, MLE, or RTE with detailed diagnostics
- FR-SB-06: Write all results to test_results.json in session workspace

Security Requirements

- NFR-SB-01: Each execution in an isolated container with no network access
- NFR-SB-02: CPU time limit ≤ 5 seconds per test case
- NFR-SB-03: Memory limit ≤ 256 MB
- NFR-SB-04: Filesystem access restricted to /tmp only

5.6 Verification Engine

Functional Requirements

- FR-VE-01: Require 100% sample test passage before proceeding to randomized tests
- FR-VE-02: Consider verification passed only if $V = \wedge_j \text{Test}_j(A) = \text{pass}$ for all j
- FR-VE-03: On failure, append structured failure record to decision_log.json with cause
- FR-VE-04: Signal orchestrator to escalate to next agent tier on any failure
- FR-VE-05: Support comparison mode for problems with multiple valid outputs

5.7 Rating Manager

Rating Formula

Each agent tier maintains an independent ELO-style rating. After each session:

Symbol	Definition
Rp	Problem rating from Codeforces
Ra	Current agent tier rating
E	Expected success: $1 / (1 + 10^{(Rp-Ra)/400})$
S	Actual outcome: 1 if solved, 0 otherwise
K	Update constant (default: 32)
R_new	$R_{old} + K(S - E)$

Functional Requirements

- FR-RM-01: Maintain separate rating per agent tier (H1–H5)

- FR-RM-02: Track solve rate per difficulty bucket (800–1000, 1000–1200, …, 2400+)
- FR-RM-03: Support promotion trigger: if Ra > R_threshold, agent is eligible for promotion
- FR-RM-04: Support demotion trigger: if Ra < R_threshold, agent scope is restricted
- FR-RM-05: Persist rating history across sessions for trend analysis

6. Cost and Computation Control

6.1 Token Budget Policy

Each agent tier enforces a strict token ceiling. If an agent exceeds its budget, it must summarize its output before returning control to the orchestrator.

Agent	Max Input Tokens	Max Output Tokens	Model Tier
H1 — Intern	2,000	500	Small
H2 — Engineer	4,000	1,000	Small
H3 — Senior	6,000	2,000	Medium
H4 — Lead	8,000	3,000	Medium
H5 — CEO	12,000	4,000	Large

6.2 Adaptive Escalation

Higher-tier agents are only invoked when:

- Confidence score falls below agent-specific threshold θi
- Verification fails after the previous agent's solution
- Complexity risk indicator is flagged in constraint_analysis.json

This ensures that the expected total cost E[Ctotal] << Cmax because high-tier agents are rare activations. Worst-case cost is bounded: Cworst = k · r · Tmax where k = active agents, r = revision cycles ≤ 2, Tmax = token budget.

6.3 Failure Controls

- Maximum 2 revision cycles per session — hard limit
- Hard termination if verification does not improve between revision cycles
- No agent may override the decision of a higher-authority agent
- Session auto-resets after any terminal state (SOLVED, UNRESOLVED, or TIMEOUT)

7. Chess Strategic Abstraction Layer

After the final solution is confirmed, the CEO agent (H5) optionally generates a chess-inspired explanation. This layer is pedagogically motivated and has no effect on the correctness pipeline.

Algorithm Paradigm	Chess Strategic Archetype
Greedy	Initiative play — seize immediate advantage
Dynamic Programming	Positional accumulation — build long-term structure
Backtracking	Variation calculation — explore branches, prune bad lines
Divide and Conquer	Piece coordination — split board into manageable sectors
Game Theory / Minimax	Perfect adversarial play — anticipate opponent moves
Graph / BFS / DFS	Piece mobility — control space and access
Binary Search	Prophylaxis — eliminate entire ranges of possibilities

Requirements

- FR-CA-01: Chess explanation is generated only after V = pass and confidence ≥ τ
- FR-CA-02: Mapping is defined in a configurable JSON file — not hardcoded
- FR-CA-03: Explanation must note that it is analogical and not a formal proof

8. Data Flow

The nominal orchestration flow for a single problem session:

1. API Gateway receives solve request with problem_id
2. Orchestrator fetches problem data and initializes /session_id/ workspace
3. Orchestrator determines active agent set $\alpha(R_p)$ from problem rating
4. Intern (H1) reads problem_spec.json; writes constraint_analysis.json and candidate_strategies.json
5. Engineer (H2) refines strategies; updates candidate_strategies.json
6. Senior (H3) writes proof_notes.md and validates complexity (if $R_p \geq 1200$)
7. Lead (H4) selects optimal strategy (if $R_p \geq 1800$); writes decision_log.json entry
8. Code is generated and written to final_solution.cpp
9. Execution Sandbox compiles and runs all test cases; writes test_results.json
10. Verification Engine checks $V = \wedge_j \text{Test}_j(A)$
11. If PASS and confidence $\geq \tau$: CEO (H5) finalizes; Rating Manager updates; session archived
12. If FAIL: escalate to next tier (max 2 cycles); if no improvement: mark UNRESOLVED
13. Session workspace is reset

9. Security and Integrity

9.1 Sandbox Security

- All code execution in isolated container with no external network access
- CPU and memory hard limits enforced at OS level
- No persistent filesystem writes outside /tmp
- Container destroyed after each session

9.2 Input Validation

- All Codeforces API inputs validated and sanitized before passing to agents
- Prompt injection mitigated via strict input parsing and schema enforcement
- Agent outputs validated against expected JSON schemas before file writes

9.3 Session Isolation

- Each session assigned a UUID — no shared state with other sessions
- Session files accessible only to the owning orchestrator instance
- Cross-session contamination detected via automated integrity checks

10. Scalability and Deployment

10.1 Microservice Decomposition

- Orchestrator Service — stateless; horizontally scalable
- Agent Pool Service — stateless workers; scale by model tier independently
- Session Memory Manager — shared storage (e.g., Redis / S3) with per-session namespacing
- Execution Sandbox — stateless containers; auto-scaled via queue depth
- Rating Manager — persistent store; single writer with read replicas

10.2 Load Balancing

- High-rated problems ($Rp > 2200$) routed to priority compute queue
- Low-rated problems processed via standard queue with lower model allocation
- Token Budget Monitor runs as sidecar; enforces limits without blocking hot path

10.3 Worst-Case Overhead

With bounded agents k, revision cycles $r \leq 2$, and per-agent token ceiling Tmax, worst-case total token cost is $Cworst = k \cdot r \cdot Tmax = O(1)$ relative to number of problems. Recursion depth does not grow unbounded.

11. Evaluation Plan

11.1 Benchmark Dataset

- Historical Codeforces problems: 500 problems per rating bucket
- Held-out unseen problem batch for final evaluation
- Comparison against single-agent LLM baseline and deterministic template solver

11.2 Ablation Studies

Each component will be individually removed and its contribution measured:

Ablation	Expected Impact
Remove agent hierarchy	Reduced solve rate on Rp > 1800; increased hallucination
Remove execution verification	False-positive solution rate increases significantly
Remove token cap	Cost per problem increases 3–5x; no quality improvement
Remove session isolation	Cross-contamination artifacts; non-reproducible results
Remove chess abstraction	No impact on correctness; reduced explanation quality

11.3 Statistical Analysis

- Solve rate comparison uses two-proportion z-test against single-agent baseline
- Significance threshold: $p < 0.05$
- Reporting: mean ± 95% CI across 5 evaluation runs per configuration

12. Future Work

- Reinforcement learning for agent promotion and demotion based on historical performance
- Self-play evaluation against historical Codeforces submission data
- Distributed execution framework for parallel stress-testing
- Support for interactive and output-only problem types
- Automated generation of difficulty-specific training data from session logs
- Public leaderboard integration with live Codeforces contests

Appendix A: Formal System Model

A.1 Problem Definition

A Codeforces problem is formally defined as:

```
P = (I, O, C, T)
```

where I = input specification, O = output specification, C = constraint set, T = tag set.

The system constructs S = (A, Π, Φ) where A = algorithm, Π = proof of correctness, Φ = complexity characterization, such that A satisfies C and produces correct O for all valid I.

A.2 Session State Machine

Session state: Σ = (F, D, V) where F = file state, D = decision log, V = verification status.

Transitions: Σ0 → Σ1 → … → Σk where Σk is terminal iff V = pass ∧ confidence > τ.

Monotonic improvement condition: ϕ(Σi+1) ≥ ϕ(Σi). If violated within rmax cycles, session terminates.

A.3 Strategy Selection

Candidate strategies S = {S1, S2, …, Sk} are ranked by:

```
E(Si) = w1·complexity + w2·runtime_empirical + w3·confidence
```

Final selection: S* = argmin E(Si) subject to correctness constraints.

End of Document — Hierarchical Multi-Agent Codeforces Solver PRD v1.0