CertPrep AI — Judge Playbook

Anticipated questions and prepared answers for the Agents League judging panel

Battle #2 — Reasoning Agents · Microsoft Agents League 2026

How to use this playbook: Each entry gives a likely judge question (in purple) followed by a model answer. The answers reference specific implementation details — code files, formula values, or screen sections — so they are verifiable. Feel free to expand with your own examples during the live demo.

Overall System & Architecture

Q: Can you walk us through your overall architecture and explain how the agents collaborate?

We build a linear pipeline of 7 agents grouped into 4 blocks.

Block 1 starts with the LearnerIntakeAgent collecting a 9-field form (student name, background, target exam, hours, weeks, etc.). Next, LearnerProfilingAgent infers 6 per-domain confidence scores and classifies the student into one of 4 experience levels. This produces a LearnerProfile Pydantic model that all downstream agents consume.

Block 1.1 runs two agents in parallel to the same profile: the LearningPathCuratorAgent maps each AI-102 domain to curated MS Learn modules (30+ in our catalogue, priority-boosted for risk domains), and StudyPlanAgent builds a week-by-week Gantt schedule using the Largest Remainder Method to fairly allocate study hours.

Block 1.2 is the returning-user path: ProgressAgent takes a mid-journey self-report and computes a composite readiness score (55% domain + 25% hours + 20% practice). It also triggers the Engagement Agent to send a weekly HTML email summary.

Block 2 is human-gated: the learner explicitly clicks 'Generate Quiz', AssessmentAgent samples from our 30-question bank weighted by exam domain percentages, and returns a scored AssessmentResult.

Block 3 is CertificationRecommendationAgent: if the quiz score is ≥70% it recommends booking the exam and suggests a next-certification path. Below 70% it generates a targeted remediation plan and loops the learner back to the LearningPathCuratorAgent for another study cycle.

A 17-rule GuardrailsPipeline sits between every block to catch bad inputs, out-of-bounds values, untrusted URLs, and duplicate IDs before they propagate.

Q: The reference architecture shows 3 sub-agents — learning path curator, study plan generator, and engagement agent. Did you implement all of them?

Yes, and we extended them.

The learning path curator is our LearningPathCuratorAgent — it maps each AI-102 domain to MS Learn modules, respects the learner's knowledge level, and applies priority boosting for risk domains. It's backed by a 30+ entry offline catalogue with real learn.microsoft.com URLs.

The study plan generator is StudyPlanAgent — it produces an actual Gantt chart (not just a list) with week-level scheduling, prerequisite alerts, and a review week at the end.

The engagement agent is embedded in ProgressAgent — it generates a self-contained HTML email report with KPI cards, nudge alerts, and a domain status table, dispatched via SMTP (or previewed in-app when no SMTP credentials are configured).

Beyond the reference, we added: LearnerIntakeAgent, LearnerProfilingAgent (live AI), ProgressAgent (mid-journey tracking), AssessmentAgent (quiz), CertificationRecommendationAgent (exam booking), and the full GuardrailsPipeline.

Reasoning & Multi-step Thinking

Q: How do your agents reason? Where does the actual 'reasoning' happen rather than just data transformation?

Reasoning happens at four distinct points:

1. LearnerProfilingAgent (in live mode) — uses Azure OpenAI gpt-4o with a JSON-schema-anchored system prompt that explicitly instructs the model to reason about the student's background, infer domain confidence scores, and classify knowledge levels. The model must justify each inference with a 1–2 sentence notes field.
2. StudyPlanAgent — mathematical reasoning via the Largest Remainder Method: it allocates study days proportional to (exam_weight × knowledge_deficit_factor), where the deficit factor amplifies time for weak/unknown domains. This is deterministic reasoning, not AI, but it is still a non-trivial multi-factor computation.
3. ProgressAgent — multi-factor reasoning: readiness = 0.55 × domain_score + 0.25 × hours_progress + 0.20 × practice_factor. The GO/NO-GO decision also inspects the count of 'critical' domains (behind by ≥2 rating points), not just the aggregate score. This catches students who are average overall but have a dangerous gap in a high-weight domain.
4. CertificationRecommendationAgent — conditional branching: it checks quiz score vs two thresholds (70% GO, 85% strong GO), then selects from different recommendations based on the target certification, generates a targeted remediation plan listing specific weak domains, and chooses 2–3 relevant next-cert suggestions from a lookup table.

Q: What reasoning pattern did you use for the study plan allocation? Could you get stuck in infinite loops?

We use the Largest Remainder Method (LRM) at day granularity.

The algorithm: each active domain gets a base allocation = (domain_weight × deficit_factor × total_days). Decimal remainders are ranked descending and one extra day is given to the highest-remainder domains until precision is achieved. This guarantees the total equals exactly the study budget with no overflows (guardrail G-10 also verifies this post-hoc).

For the remediation loop question: no, we cannot get into a true infinite loop because each loop iteration runs a fresh quiz, and statistically a student's knowledge improves with each study cycle. However, in the current mock mode, the profiling scores don't change between iterations. In a production live-AI deployment, we would re-profile after each study cycle to detect actual knowledge improvement. This is called out as a known extension point in our documentation.

Guardrails & Responsible AI

Q: What guardrails have you implemented and why did you choose those specific rules?

We have 17 guardrail rules in 5 categories, implemented in guardrails.py.

Input guardrails (G-01 to G-05) prevent obviously broken data from flowing to expensive AI operations — for example, we block empty student names (G-01) before calling Azure OpenAI. G-05 adds a PII transparency notice since we store the student's real name in session state.

Profile guardrails (G-06 to G-08) ensure the LearnerProfile output is structurally complete. G-07 is the most important: it BLOCKS any confidence score outside [0, 1] because downstream agents perform arithmetic on these values — an out-of-range score would silently corrupt the Gantt allocation and the readiness formula.

Plan guardrails (G-09 to G-10): G-09 blocks any study task where start_week > end_week, which was a real bug we hit during development when the LRM rounding overflowed in 3-week plans with 5 active domains. G-10 warns if allocated hours exceed the student's declared budget by more than 10%.

Progress guardrails (G-11 to G-13) validate that self-rated data is sensible before the readiness formula is applied.

Content guardrails (G-16 to G-17): G-17 is important for Responsible AI — it verifies that every URL from LearningPathCuratorAgent and CertificationRecommendationAgent originates from learn.microsoft.com, pearsonvue.com, or other trusted domains. This prevents hallucinated or injected URLs from reaching the user.

Q: How do you handle PII and data privacy in your system?

In its current form, CertPrep AI does not persist any data outside the user's browser session.

All profile data is stored in Streamlit's session_state, which is ephemeral and scoped to a single browser session. Nothing is written to a database or sent to an external service in mock mode. Guardrail G-05 explicitly shows the user a notice that their name is session-only and not transmitted externally.

In live Azure OpenAI mode, the student's background_text and existing certs are sent to Azure OpenAI for profiling. We document this in the sidebar ('Live Azure OpenAI' mode label) and recommend that users avoid including sensitive PII in the background description.

For the email feature, we only send the report to the email address the user explicitly enters; we do not store it beyond the session.

In a production version we would add: data minimisation (strip PII before LLM calls), Azure Content Safety API integration for the output content filter, and an explicit consent checkbox aligned with GDPR Article 6.

Technical Implementation

Q: How does the study plan allocation algorithm work? Why did you choose it?

We use the Largest Remainder Method (LRM) at day granularity — the same algorithm used in proportional election systems to distribute seats fairly.

Step 1: Compute each domain's raw day allocation = total_days × (exam_weight × deficit_factor). deficit_factor is 2.0 for UNKNOWN domains, 1.5 for WEAK, 1.0 for MODERATE, and 0.3 for STRONG.

Step 2: Floor each allocation to get integer days. Since floors always under-count, sum them and compute the shortfall.

Step 3: Sort domains by their decimal remainders descending and give one extra day to the top-N domains until we've distributed exactly total_days.

Step 4: Convert days back to week ranges (start_week = 1 + cumulative_days // 7). Working at day granularity prevents start > end bugs that occur when you have more active domains than study weeks — a real edge case we hit with 5 domains in a 3-week plan.

We chose LRM over a simpler round-robin because it correctly handles the case where one domain's weight is so small (e.g. 10%) that it rounds to zero days in a short plan.

Q: How does the readiness scoring formula work and how did you calibrate the weights?

The formula is: readiness_pct = 0.55 × weighted_domain_score + 0.25 × hours_progress_pct + 0.20 × practice_factor.

weighted_domain_score: each domain's self-rating (1–5) is normalised to [0,1] and multiplied by the AI-102 exam blueprint weight. This means a poor score in Computer Vision (22.5% weight) hurts more than a poor score in Conversational AI (10% weight), matching the exam's stakes.

hours_progress_pct: simple ratio of hours_spent to total_budget_hours, capped at 100%. This rewards consistent study effort.

practice_factor: 'yes' + score ≥ 70% gives 1.0; 'yes' + score < 70% gives score/100 (partial credit); 'some' gives 0.5; 'no' gives 0.2 (we do not penalise heavily since early in prep practice exams aren't expected).

> Weight calibration rationale: domain knowledge is the strongest predictor of exam success (55%), time invested matters but not as much (25%), and practice exams are a leading indicator but students shouldn't be penalised for not having done them yet (20%). These weights are inspired by published Microsoft certification coaching guidance.

Q: The reference suggests integrating with the Microsoft Learn MCP server. Did you use it?

In our current implementation we use a hardcoded offline catalogue of 30+ MS Learn module URLs (all real, verified learn.microsoft.com pages) rather than calling the live MCP server. This was a deliberate trade-off for reliability and demo stability — the app works completely offline without any external API dependency.

Architecturally, the LearningPathCuratorAgent is designed to be swapped: the _LEARN_CATALOGUE dict in learning_path_curator.py can be replaced by a call to the Microsoft Learn Catalog API (https://learn.microsoft.com/api/catalog/) or the MCP server without changing any downstream code, because the output shape (LearningPath dataclass) remains identical.

The MCP integration would give us real-time module availability, updated durations, and new content that post-dates our catalogue. We'd add it as enhancement Option A in any production roadmap.

Q: What happens if Azure OpenAI is unavailable or the API key is wrong?

We have a three-layer fallback strategy.

Layer 1 (before the call): Guardrail G-04 checks that the exam code is valid, and if Azure credentials aren't loaded, the sidebar shows a warning and automatically switches back to Mock mode (we check if az_endpoint and az_key are both non-empty before allowing Live mode).

Layer 2 (during the call): the LearnerProfilingAgent call is wrapped in a try/except block. If it throws (AuthenticationError, RateLimitError, network failure, etc.), we display a Streamlit error with the exception message and then fall back to run_mock_profiling(), setting mode_badge to '■ Mock (fallback)' so the user knows they're not getting a live AI result.

Layer 3 (post-hoc): Profile guardrails G-06/G-07/G-08 run on whatever profile was returned (mock or live) to ensure structural validity before downstream agents receive it.

This means the app is always usable, even in a demo environment with no Azure credentials.

User Experience & Demo

Q: Can you demo the end-to-end flow right now? What should we look for?

Absolutely. Here's the suggested demo script:

1. Open http://localhost:8501 in a browser.

2. Fill in the intake form as 'Priya Sharma', background 'Python developer with 3 years experience, familiar with REST APIs and Azure', exam AI-102, 10 hrs/week, 8 weeks, concerns: 'Azure OpenAI RAG, Bot Service'. Click 'Generate Learner Profile'.

3. Notice the 5 coloured KPI cards: Student (purple), Experience (blue), Study Budget (green), Risk Domains (orange), Avg Confidence (dynamic).

4. Go to Domain Map tab — point out the radar chart and the Radar Insights callout showing the strongest axis, below-threshold domains, and the bar chart insights.

5. Go to Study Setup tab — show the prerequisite banners (AI-900 recommended), then the Gantt chart with colour-coded priority bars and Week N labels. Hover over a bar to see the tooltip.

6. Go to Learning Path tab — expand a risk domain like Generative AI to show the MS Learn module links with difficulty/duration tags.

7. Go to My Progress tab — enter 20 hours, 4 weeks elapsed, rate domains with sliders, click 'Assess My Readiness'. Show the gauge, GO/NO-GO card, and nudge alerts.

8. Go to Knowledge Check tab — click 'Generate New Quiz', answer a few questions, submit, show the domain breakdown and per-question feedback.

9. Go back to Recommendations tab — the CertificationRecommendationAgent now shows its output (GO/NO-GO + exam logistics + next cert suggestions).

Q: How does the system handle a returning user — someone who comes back after a week of studying?

The returning-user journey is designed through the sidebar radio button '■ Returning — update my progress'.

Selecting it changes the hero banner to a personalised welcome-back message using the stored session_state profile name and last check-in readiness score.

In the My Progress tab, if a previous ProgressSnapshot exists in session_state, all form fields are pre-populated with the last values: hours, weeks, domain sliders, and practice exam status. The top of the tab shows a 'Last check-in result' card with the previous readiness score, verdict, and GO/NO-GO in the appropriate colour.

When the learner re-submits, the new assessment overwrites the previous one. Smart nudges compare current vs expected progress — for example, if the student is behind on hours or still hasn't done a practice exam, a WARNING-level nudge fires.

The weekly email sender is also available here: if the user entered their email in the sidebar, it's pre-filled in the 'Send report to' field.

Extensibility & Future Work

Q: What would you add if you had another week?

In priority order:

1. Live Microsoft Learn MCP server integration — replace the offline catalogue with real-time data from https://github.com/microsoftdocs/mcp so module availability and durations stay current.

2. Azure AI Foundry orchestration — deploy the 7 agents as Azure AI Foundry Agent Service endpoints with the Foundry SDK, enabling proper telemetry, token usage tracking, and cloud-hosted evaluation.

3. Persistent learner storage — replace session_state with Azure Cosmos DB so a learner can close the browser and return to exactly where they left off across weeks of preparation.

4. Azure Content Safety API — replace the heuristic G-16 keyword filter with the real Content Safety service (harm categories: hate, self-harm, sexual, violence).

5. Adaptive quiz — instead of random sampling from the static bank, use Item Response Theory to select questions slightly above the learner's current ability for maximal learning signal (computerised adaptive testing).

6. Evaluation pipeline — add test cases for each agent using the Microsoft Foundry SDK evaluation tools, scoring accuracy, relevance, and faithfulness with automated metrics like BLEU/ROUGE for text outputs.

Q: Is this system limited to AI-102? How would you extend it to other certifications?

AI-102 is the primary target but the architecture is certification-agnostic.

The domain registry (AI102_DOMAINS in models.py) is the only AI-102-specific data structure. To support DP-100, AZ-204, or any other certification we would:

1. Add a new domain list (e.g. DP100_DOMAINS) with the corresponding syllabus areas and weights. 2. Extend _LEARN_CATALOGUE in learning_path_curator.py with domain-to-module mappings for the new cert. 3. Add questions for the new domains to _QUESTION_BANK in assessment_agent.py. 4. Add prerequisite entries to _CERT_PREREQ_MAP in study_plan_agent.py. 5. Add next-cert entries to _NEXT_CERT_MAP in cert_recommendation_agent.py.

The LearnerProfilingAgent prompt (in live mode) already includes the exam_target in its system message, so it naturally adapts to a different certification. The guardrails, scoring formula, and UI are all target-cert-independent. Realistically, we could support a new certification with 1–2 hours of data entry.

Quick Reference — Key Numbers & Files

Item	Value / Location
Total agents	7 (Intake, Profiling, Curator, Planner, Progress, Assessment, CertRec)
Total guardrails	17 rules across 5 categories (BLOCK / WARN / INFO)
Assessment question bank	30 questions across 6 domains × 3 difficulty levels
Readiness formula	0.55 × domain + 0.25 × hours + 0.20 × practice
GO threshold	≥ 75% readiness + 0 critical domains
Assessment pass mark	≥ 60% quiz score → feeds CertRec agent
Study plan algorithm	Largest Remainder Method at day granularity (7 days/week)
UI tabs	7 tabs: Domain Map, Study Setup, Learning Path, Recommendations, My Progress, Knowledge Check, Raw JSON
Mock mode	Fully functional offline — no Azure credentials required
Live mode fallback	Azure OpenAI error → auto-fallback to mock profiler
Main UI file	streamlit_app.py (~1600 lines)
Guardrails file	src/cert_prep/guardrails.py
Documentation	docs/technical_documentation.pdf (this script's sibling output)