CertPrep Multi-Agent System — Agents League Battle #2

Prepared: 22 February 2026 · Author: Athiq Ahmed · Version 1.0

What this document covers
A complete audit of every Azure AI / Microsoft service called, mocked, or planned in the CertPrep multi-agent Streamlit application. For each service the report maps: which agent/block uses it, why it is used, how it is called, and a personal recommendation on whether the current usage is appropriate — or should be replaced, deferred, or cached for a production deployment.

1. Executive Summary

The CertPrep system is a multi-agent AI application built for the Microsoft Agents League hackathon. It orchestrates eight specialised agents to create personalised Microsoft certification study plans. Despite being marketed as an 'AI-powered' system, the vast majority of its logic is intentionally **rule-based and deterministic** — calling Azure OpenAI only at one well-defined point in the pipeline. This is by design: determinism, testability, and zero-cost mock operation are primary goals for a demo system.

Service / Technology	Is it used?	Number of call points	Can run without it?
Azure OpenAI GPT-4o	Optional (live mode only)	1 — LearnerProfilingAgent	■ Yes — full mock mode
Microsoft Learn API	■ Not called at runtime	0 — data is hardcoded	■ Always — offline catalogue
Azure AI Content Safety	■ Not integrated yet	0 — heuristic only (G-16)	■ Yes — keyword heuristic
Azure Communication Services	■ Not integrated	0 — SMTP only	■ Yes — SMTP optional
Azure AI Search / Cognitive Search	■ Not integrated	0 — question bank hardcoded	■ Yes — in-memory bank
SQLite (local persistence)	■ Always on	1 — database.py (all CRUD)	■ Yes — optional skip mode
SMTP (Python stdlib)	Optional (env vars gated)	1 — progress summary email	■ Yes — silently skipped
Plotly + Streamlit (UI)	■ Always on	Many — charts / tabs / forms	N/A — core UI layer

2. Azure OpenAI GPT-4o

2.1 What is it?

Azure OpenAI is a managed deployment of OpenAI's GPT-4o large language model hosted on Microsoft Azure. It provides chat completions with JSON-mode structured output, meaning the model can be instructed to return a strict JSON schema — used here to generate a typed **LearnerProfile** Pydantic object.

2.2 Where is it used?

Block	Agent/Module	Streamlit Tab	Call	Mode
B0 — Learner Profiling	LearnerProfilingAgent (b0_intake_agent.py)	Tab 1 — Intake & Setup	`chat.completions.create() GPT-4o · json_object mode temp=0.2 · max_tokens=2000`	Optional (use_live flag)

2.3 How it is called

The **LearnerProfilingAgent** sends a structured system prompt (containing the full JSON schema for *LearnerProfile*) and a user message containing all intake form fields. GPT-4o returns a JSON object that is parsed and validated by Pydantic. If validation fails, a *ValidationError* is raised before any downstream agent runs. The call uses **temperature=0.2** to maximise determinism (low creative variation).

System prompt pattern: 'You are an AI certification advisor. Given the student background, return ONLY valid JSON matching this schema…' — no prose, no explanation, just structured data. This is the most appropriate use of an LLM: transforming unstructured free-text input into a validated, typed data contract.

2.4 Mock mode — no Azure OpenAI needed

When use_live = False (the default), **b1_mock_profiler.py** is used instead. The mock profiler is a rule-based Python module (no API calls) that applies regex keyword matching, experience level inference, cert domain boost matrices, and concern topic mapping to produce an identical *LearnerProfile* output contract. For the five demo personas, mock output is indistinguishable from live GPT-4o output.

2.5 Recommendation

■ KEEP — but gate it strictly

Using Azure OpenAI at this one point (intake profiling) is the correct architectural decision. The LLM is being used to solve a real NLP problem — interpreting unstructured background text, mapping it to structured domain confidence scores, and inferring learning style and experience level. A rule-based mock can approximate this for five known personas, but for arbitrary real-world students the LLM provides meaningfully better profiling.

What to avoid: Do NOT call Azure OpenAI from StudyPlanAgent, AssessmentAgent, ProgressAgent, or CertRecommendationAgent. These agents do deterministic computation (Largest Remainder allocation, weighted formula, rule-based routing) where an LLM adds latency and cost with no accuracy benefit. An LLM 'recommending' which week to study a domain is worse than a deterministic algorithm that uses actual exam weightings.

Production improvement: Add a response cache keyed by a hash of the intake inputs. If two students have near-identical backgrounds/certs/concerns, return the cached profile rather than paying for a second GPT-4o call. Use functools.lru_cache or Redis.

3. Microsoft Learn Module Catalogue

3.1 What is it?

Microsoft Learn (*learn.microsoft.com*) provides a public REST API that lists all available training modules by exam or topic. In principle the app could call GET https://learn.microsoft.com/api/catalog/?locale=en-us&type;=modules at runtime to dynamically fetch the latest module list for each domain.

3.2 Current usage — offline / hardcoded

The current implementation in **b1_1_learning_path_curator.py** does NOT call the MS Learn API. Instead, it uses a Python dictionary (_LEARN_CATALOGUE) with manually curated module metadata for five exam families (AI-102, AI-900, DP-100, AZ-204, AZ-305). This includes title, direct URL, duration in minutes, difficulty, type (module/path), and display priority (core / supplemental / optional).

Exam	Domains covered	Module count	Data source
AI-102	6 domains	~36 modules	Hardcoded dict
AI-900	5 domains	~20 modules	Hardcoded dict
DP-100	6 domains	~24 modules	Hardcoded dict
AZ-204	5 domains	~18 modules	Hardcoded dict
AZ-305	4 domains	~16 modules	Hardcoded dict

3.3 Recommendation

■ KEEP AS HARDCODED — correct for demo scale; plan API integration for production

For a hackathon demo covering five known exam families, live MS Learn API calls add latency, internet dependency, rate-limit risk, and API contract drift risk — for zero learner benefit. The curated catalogue is faster, fully offline, always consistent, and lets you control exactly which modules are shown (human editorial quality gate). Keep it.

For production deployment: Schedule a nightly *Azure Function* (or GitHub Action) that calls the MS Learn Catalog API, diffs against the stored catalogue, and updates only changed/new modules. The Streamlit app always reads from the locally cached version — never hitting the API at request time. This gives you live data freshness without per-request API latency.

Do NOT call the MS Learn API on every page load or tab switch. The catalogue changes at most weekly. Calling it per-request would add 500ms–2s per user, risk rate-limiting, and produce no better output than the curated offline data.

4. Azure AI Content Safety

4.1 What is it?

Azure AI Content Safety is a managed API that classifies text and images against four harm categories (Hate, Self-Harm, Sexual, Violence) with severity scoring, and provides a Prompt Shield feature specifically designed to detect prompt injection attacks against LLM-based systems.

4.2 Current usage — heuristic only (G-16)

The current **GuardrailsPipeline** rule **G-16** implements a basic keyword scan over free-text intake fields (background, concern_topics, goal_text) looking for heuristic patterns (expletives, violence keywords). This is a pure Python check with no external API call. It is intentionally conservative — it rarely fires in practice since the user population is certification students.

4.3 Recommendation

■ UPGRADE — replace G-16 with Azure AI Content Safety API in production

The heuristic keyword scan (G-16) is sufficient for the hackathon demo. A real deployment accepting arbitrary student text from the public should use Azure AI Content Safety for G-16 to get supported, maintained, multi-language harm detection with severity levels.

Recommended call pattern: One Content Safety API call per intake form submission (not per keypress, not per tab change). Call it asynchronously alongside the profiler call — not before it — so the 200–400ms latency is hidden in the parallel execution. Only BLOCK-level harms halt the pipeline; LOW severity returns a WARN.

Do NOT call Content Safety on every agent transition or on the study plan output. The system generates only structured data (not freehand LLM prose) after the intake, so there is no untrusted text to scan downstream. Over-calling adds latency and cost with no safety benefit.

5. Assessment Question Bank & SQLite Persistence

5.1 Assessment Agent — Hardcoded Question Bank

The **AssessmentAgent** (b2_assessment_agent.py) maintains a 30-question bank per exam family, manually authored and stored in Python data structures. The quiz sampling algorithm (Largest Remainder Method) draws a weighted subset of questions proportional to real exam domain weights. **No external API is called at any point in the quiz lifecycle.**

Exam	Questions in bank	Source	Serving method
AI-102	30 (5 per domain)	Manually authored	In-memory random.sample()
DP-100	30 (5 per domain)	Manually authored	In-memory random.sample()
AI-900 + others	Shared bank	Manually authored	In-memory random.sample()

5.2 SQLite — Session Persistence

All student, profile, plan, learning path, trace, and progress data is persisted to cert_prep_data.db via Python's standard-library **sqlite3** module (wrapped in **database.py**). This enables session recovery on page refresh: a returning student re-enters name + PIN to restore their full profile and plan.

5.3 Recommendations

Component	Current	Demo verdict	Production upgrade
Question bank	30 hardcoded Qs per exam	■ Keep — no API needed for 5 known exams	Azure AI Search index — allow dynamic Q bank growth, semantic search for relevant questions
SQLite	Local file, zero deps	■ Keep — correct for demo	Azure Cosmos DB (NoSQL, serverless) for multi-user, multi-region, auto-scale

6. Email / SMTP Notification

6.1 What is it?

The **ProgressAgent** (b1_2_progress_agent.py) includes an optional email dispatch path. After generating a weekly study summary, the function attempt_send_email() reads SMTP configuration from environment variables (SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASS) and sends an HTML email using Python's smtplib stdlib. If the SMTP env vars are absent, the function silently returns False — the pipeline continues without error.

6.2 Current usage in Streamlit

In the Streamlit UI, the 'Send Weekly Summary' button in the Progress tab calls generate_weekly_summary() to produce an HTML report, then calls attempt_send_email() with the student's email address from session state. The email is sent only if SMTP credentials are set — default demo mode shows the report in the UI without sending.

6.3 Recommendation

■ ACCEPTABLE for demo — replace with Azure Communication Services for production

SMTP via Python stdlib works and has zero external dependency on Azure. For a demo that may never send a real email, this is fine. The silent-fail pattern (returns False, pipeline continues) is good defensive design.

Production upgrade: Replace smtplib with Azure Communication Services Email SDK. Benefits: guaranteed delivery tracking, bounce/unsubscribe handling, template rendering, no SMTP port blocking on Azure-hosted VMs, and full audit log in Azure Monitor. The code change is isolated to attempt_send_email() only.

Do NOT send an email on every tab navigation or progress form save. Email dispatch should be explicit user action (button click) only, as currently implemented.

7. Agents That Intentionally Use No External Service

The following agents are entirely rule-based or algorithmic. They make no network calls, require no API keys, and run at sub-millisecond speed. This is intentional — these decisions are deterministic and should remain so.

Agent	Module	Algorithm	Why no LLM

Study Plan Generator	b1_1_study_plan_agent.py	Largest Remainder Method, risk-sorted task scheduling, prereq gap detection	Domain weight allocation is deterministic math. An LLM 'allocating weeks' would be less accurate than using official exam weights.
Progress Tracker	b1_2_progress_agent.py	Weighted formula: 0.55×conf + 0.25×hours + 0.20×practice	The readiness formula is transparent & user-auditable. An LLM producing a readiness score would be a black box.
Assessment Builder & Scorer	b2_assessment_agent.py	Weighted random sampling from hardcoded Q bank, exact-match scoring	Quiz scoring is binary correct/incorrect. GPT-4o generating questions risks hallucinated answers and inconsistent difficulty.
Cert Recommender	b3_cert_recommendation_agent.py	Rule-based next-cert path matching from cert chain table	Certification paths are fixed Microsoft exam chains. Rules are more reliable than an LLM that might suggest non-existent certifications.
Guardrails Pipeline	guardrails.py	17 deterministic validation rules (BLOCK/WARN/INFO)	Safety rules must be auditable and testable. An LLM-based safety check is non-deterministic and cannot be reliably unit tested.

Key principle: Use an LLM only where the task is genuinely language-understanding — interpreting ambiguous free text, inferring intent, or generating natural language output. Never use a LLM to replace arithmetic, sorting, lookup tables, or boolean logic.

8. Master Recommendation Table

This table consolidates all recommendations across every service touchpoint in the system. Use this as a production readiness checklist.

Service / Pattern	Current	Verdict	Recommendation
Azure OpenAI GPT-4o (LearnerProfilingAgent)	Optional — live mode only	■ KEEP	Best use of LLM in the system. Adds structured profile from free-text. Add response caching for identical inputs. Use structured output + low temperature.
Azure OpenAI GPT-4o (all other agents)	NOT used — correctly	■ KEEP AS-IS	Do NOT add OpenAI calls to StudyPlan, Assessment, Progress, or CertRec agents. These are deterministic computations. LLM adds latency + cost + non-determinism.
MS Learn Catalog API (LearningPathCurator)	NOT called — hardcoded offline catalogue	■ KEEP for demo ■ Upgrade for prod	Demo: keep hardcoded. Production: schedule nightly sync (Azure Function / GitHub Action). Never call per-request.

Azure AI Content Safety (Guardrail G-16)	Heuristic keyword scan — no API	■ UPGRADE for prod	Replace G-16 keyword heuristic with Azure AI Content Safety API in production. Call once per form submit, asynchronously. Never call per tab load.
Azure AI Search (Assessment Q bank)	NOT used — in-memory bank	■ KEEP for demo ■ Upgrade for prod	Demo: in-memory bank is fine for 5 exams × 30 questions. Production: Azure AI Search index for semantic question retrieval + larger bank.
SQLite (Session persistence)	Always on — local file	■ KEEP for demo ■ Migrate for prod	Demo: SQLite is ideal (zero deps, portable). Production: Azure Cosmos DB for multi-user, multi-instance, serverless scale.
SMTP (Progress email)	Optional — env-var gated, silent-fail	■ KEEP for demo ■ Upgrade for prod	Demo: SMTP stdlib is fine. Production: Azure Communication Services Email SDK for delivery tracking, bounces, unsubscribe compliance.
Observability (AgentStep/RunTrace)	SQLite + Admin Dashboard	■ KEEP for demo ■ Upgrade for prod	Demo: custom trace structs + Admin Dashboard are excellent for hackathon. Production: Azure Application Insights + Log Analytics Workspace.
Authentication (PIN-based login)	Name + 4-digit PIN, SHA-256 stored	■ Fine for demo ■ Must change for prod	Demo: PIN auth is fine. Production: Azure AD B2C / Entra External ID. Never ship PIN auth to real users.
Secrets management (.env file)	.env file / env vars	■ Fine for demo ■ Must change for prod	Demo: .env is standard. Production: Azure Key Vault + Managed Identity. No secrets in code or env vars on shared infrastructure.

9. Decision Framework — When to Use Azure AI Services

Use the following criteria to evaluate whether any new Azure AI service call is justified:

Question	If YES →	If NO →
Is the input genuinely unstructured free text that requires language understanding?	■ Azure OpenAI may be appropriate	■ Use a rule/algorithm instead
Would two identical inputs always produce the same correct output?	■ Use deterministic code — no LLM needed	■ LLM or probabilistic model may help
Can the data be safely cached between requests?	■ Always cache — avoid redundant API calls	Consider if real-time freshness is truly required
Does the data change more often than daily?	Evaluate real-time API need vs refresh schedule	■ Batch-sync nightly — never per-request
Will the user wait for this API call on a demo?	■ Minimise — mock, cache, or async	■ Background call acceptable
Is the output safety-critical (user-facing verdict, booking advice)?	■ Use deterministic formula — explainable by design	LLM-generated verdict is acceptable for non-critical guidance

The Anti-Pattern to Avoid

■ Do not use Azure OpenAI as a general-purpose function replacement

There is a common over-engineering trap where every data transformation, recommendation, or output in an app is routed through an LLM — 'because it's an AI app'. This produces: higher latency (2–5s per call), higher cost, non-deterministic outputs, untestable logic, and worse results than simple rules for structured computations. In this system, the study plan algorithm, readiness formula, quiz sampling, and cert routing are all more accurate, faster, and cheaper as deterministic code.

The right mental model: The LLM is a natural-language-to-structured-data converter at the boundary between the messy human world and the clean typed world of the pipeline. Once data is typed and structured, keep all computation deterministic.

10. Production Services Roadmap

Ordered by impact — highest priority first. Each item is an independent upgrade requiring no changes to the orchestration pipeline or agent contracts.

Priority	Upgrade	Replaces	Effort	Impact
1 — HIGH	asyncio.gather() for StudyPlan + LearningPath agents	Sequential execution	~0.5 days	Saves ~3s per user (10→7s live mode latency)
2 — HIGH	Azure AI Content Safety API (G-16)	Keyword heuristic scan	~1 day	Supported harm detection; multi-language; prompt shield
3 — HIGH	OpenAI response cache (Redis or lru_cache)	Uncached GPT-4o calls	~1 day	Eliminates redundant API cost for returning students or near-identical profiles
4 — MEDIUM	Nightly MS Learn catalogue sync (GitHub Action)	Static hardcoded dict	~2 days	Learning path stays current without per-request API calls
5 — MEDIUM	Azure Cosmos DB NoSQL (serverless)	SQLite local file	~2 days	Multi-user, multi-region persistence; handles 1000+ concurrent users
6 — MEDIUM	Azure Communication Services Email	SMTP stdlib	~1 day	Delivery tracking, bounce handling, compliance — isolated to attempt_send_email()
7 — MEDIUM	Azure AD B2C authentication	PIN-based login	~3 days	Real identity, SSO, MFA — required before any public launch
8 — LOW	Azure Application Insights	SQLite trace + Admin Dashboard	~1 day	Structured telemetry, per-agent latency dashboards, alerting
9 — LOW	Azure AI Search — dynamic Q bank	Hardcoded 30-Q bank	~5 days	Enables semantic Q retrieval, larger bank, faster updates — only needed at scale

10 — FUTURE	Magentic-One multi-expert profiler deliberation	Single-agent profiling	~1 week	Multiple domain-expert agents debate learner skill level; better for edge-case profiles