GoalForge AI: A Seven-Agent Orchestration Framework for Autonomous and Adaptive Study Planning

S.THASLIMA FARJANA
PG Department of Computer Science, Syed Ammal Arts and Science College,
Ramanathapuram, India

Abstract
Traditional study planning tools function as static schedulers that neither adapt to a learner's progress nor explain their recommendations. This paper presents GoalForge AI, an agentic study-planning system built on a seven-agent orchestration architecture — Planner, Research, Decision, Execution, Validation, Memory, and Reporting — where each agent handles one stage of the planning lifecycle and every stage is logged for full traceability. The system is implemented using Node.js, React, SQLite, and the Gemini API. A deterministic, rulebased Decision agent selects a pacing strategy from milestone density, keeping this critical step explainable and independent of external API availability. Functional and integration testing confirm the seven-agent pipeline executes correctly end-to-end, the Decision agent responds correctly to changing density, and the system degrades gracefully via fallback when the AI service is unavailable. Results demonstrate that decomposing agentic planning into specialized, auditable agents is a practical approach to explainable AI in personalized education.

Keywords—Agentic AI, Multi-Agent Orchestration, Personalized Learning, Explainable AI, Autonomous Planning.

I. INTRODUCTION

Artificial Intelligence has progressed from systems that execute single, narrowly defined tasks to agentic systems capable of understanding objectives, planning multi-step actions, and adapting to changing conditions with limited human intervention. Agentic AI represents a shift in emphasis from prediction — producing a single response to a single prompt — toward autonomous execution, in which a system decomposes a goal into a sequence of actions, executes them, observes the outcome, and revises its plan accordingly. In education, this shift creates an opportunity to move beyond static digital timetables toward planning tools that reason about a learner's goals, monitor progress continuously, and revise recommendations autonomously rather than requiring the learner to manually re-plan. This opportunity is particularly relevant in academic settings where learners routinely juggle multiple simultaneous goals — coursework, competitive examinations, and skill development — each with its own deadline and pace, a coordination problem that static tools were never designed to solve.

Students preparing for examinations, certifications, or long-term learning goals commonly rely on manually created timetables or generic scheduling applications. A review of widely used tools reveals three recurring limitations. First, such tools exhibit static structure: once a timetable is created it does not adjust when a learner falls behind, completes work ahead of schedule, or the deadline itself changes. Second, AIassisted planners that route every decision through a single large language model call exhibit opaque reasoning — the justification for a particular schedule cannot be isolated, inspected, or selectively corrected. Third, most tools provide no persistent personalization, failing to retain learner-specific context such as a preferred pacing strategy across multiple sessions or goals.

This paper introduces GoalForge AI, a study-planning application that operationalizes agentic AI through a sevenagent orchestration architecture. Rather than routing every decision through a single opaque model call, GoalForge AI decomposes the planning problem into seven specialized responsibilities: goal decomposition, resource research, strategic decision-making, task execution, progress validation, long-term memory, and human-readable reporting. Each responsibility is assigned to a dedicated, independently testable agent module, and every agent's input and output is written to a persistent audit log, allowing a plan's provenance to be inspected stage by stage — a property this paper refers to as agent-level traceability. This traceability is the central contribution of the work: it distinguishes GoalForge AI from single-model planners in which the reasoning behind a recommendation cannot be separated from the recommendation itself.

The remainder of this paper is organized as follows. Section II reviews related work in conventional and AIassisted planning tools as well as multi-agent orchestration. Section III describes the proposed methodology, including the seven-agent architecture, the system workflow, and implementation details. Section IV presents the experimental setup, verification results, and discussion. Section V concludes the paper and outlines directions for future work.

The contribution of this work is threefold. First, it presents a concrete seven-agent decomposition of the study-planning task, with each agent's responsibility, input, and output explicitly defined. Second, it demonstrates that mixing a deterministic, rule-based agent (Decision) with four LLMdriven agents (Planner, Research, Execution's supporting research, and Reporting) within a single pipeline is both technically straightforward and materially improves explainability at the specific step — pacing strategy selection — where predictability matters most to a learner. Third, it documents a fault-tolerant design pattern in which every LLM-backed agent degrades gracefully to a deterministic fallback, ensuring the full pipeline, database layer, and user interface remain testable and demonstrable independent of third-party API availability, a practical concern for academic and early-stage deployments operating on free-tier API quotas.

II. RELATED WORK

Conventional planning tools such as digital calendars, todo list applications, and static timetable generators allow manual task entry but provide no automated decomposition of a learning goal into milestones, nor do they adapt scheduling based on progress. The learner bears the full cognitive load of breaking a goal into actionable steps and of re-planning whenever circumstances change, which is precisely the burden an agentic system is well suited to absorb.

More recent AI-assisted planning tools typically wrap a single large language model prompt around scheduling logic: the entire plan is generated from one model call given the goal and the deadline. This design improves flexibility relative to static tools, since the model can generate varied schedules for varied inputs, but it sacrifices explainability. The reasoning behind a specific recommendation cannot be isolated, audited, or selectively corrected without regenerating the plan in its entirety, which limits both learner trust and a developer's ability to debug incorrect output [1].

Outside the education domain, multi-agent orchestration frameworks — in which distinct agents handle planning, retrieval, execution, and evaluation as separate components — have been shown to improve both interpretability and robustness to individual component failure, since a fault in one agent does not necessarily corrupt the reasoning of others [2], [3]. Large language models themselves have demonstrated strong few-shot reasoning capability that makes them suitable as the reasoning core for individual agents within such a pipeline, provided their output is constrained and validated by the surrounding system [2]. Prior explainable-AI literature has also argued that decomposing a decision process into inspectable sub-steps is often more valuable to an end user than post-hoc explanation of an already-opaque model, which motivates this paper's emphasis on architectural transparency over explanation generation as an afterthought [1], [3]. GoalForge AI applies this multi-agent principle specifically to the personalized education domain, an application area where existing literature and commercial tools have relied largely on either static rule-based scheduling or a single end-to-end model, leaving a gap that this work addresses through an explicit, auditable seven-agent decomposition.

TABLE I. COMPARISON WITH EXISTING PLANNING APPROACHES

Approach	Adapts	Explainable	Persistent
Static calendar / to-do app	No	N/A	No
Single-model AI planner	Partial	No	Rarely
GoalForge AI (proposed)	Yes	Yes	Yes

III. PROPOSED METHODOLOGY

A. System Overview

GoalForge AI is a full-stack web application in which a learner registers, defines a study goal together with a deadline, and receives an automatically generated, day-byday task plan. Internally, goal creation triggers a sequential pipeline of seven agents that together decompose the goal, gather contextual study material, choose a pacing strategy, generate concrete tasks, validate ongoing progress, persist learner preferences, and produce a human-readable summary, as illustrated in Fig. 1. Every agent's input and output is logged to an audit table, allowing the complete reasoning chain behind any generated plan to be inspected after the fact. This design deliberately favors a small number of narrowly scoped agents over either a single large agent or a very large number of micro-agents: seven was found sufficient to cover the distinct concerns identified in Section I — decomposition, context-gathering, strategic choice, materialization, monitoring, personalization, and communication — without introducing excessive inter-agent coordination overhead.

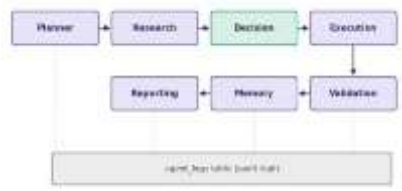


Fig. 1. Seven-agent orchestration pipeline; every stage writes its input and output to the shared agent_logs table.

B. Seven-Agent Architecture

Fig. 1 shows the seven agents grouped into two categories by how their output is produced: four agents (Planner, Research, Execution, Reporting) rely at least in part on the Gemini API for natural-language reasoning, while the Decision agent is entirely rule-based, and the Execution and Validation agents perform deterministic data transformations over already-generated content. This mixed composition is intentional: it allows the system to use a language model exactly where open-ended reasoning is genuinely required —

decomposing an arbitrary goal description, suggesting relevant subtopics, and phrasing an encouraging summary — while keeping the decision that most directly affects the learner's day-to-day schedule fully transparent and independent of model variability.

1) Planner Agent: The Planner agent receives the learner's goal title, free-text description, and deadline in days, and constructs a prompt instructing the Gemini API to decompose the goal into four to six milestones distributed proportionally across the available time window. The response is requested in a strict JSON format so that downstream agents can consume it programmatically; if the API call fails or returns malformed output, the agent falls back to a proportionally spaced default milestone set, ensuring the pipeline never stalls on a malformed response. The milestone count itself is not fixed by configuration but emerges from the model's interpretation of the goal's scope, which the Decision agent later normalizes against the deadline through the density calculation described below.

2) Research Agent: For each milestone produced by the Planner, the Research agent issues a separate, narrowly scoped prompt asking for three concise subtopics or resource types relevant to that specific milestone. This one-milestoneat-a-time design keeps each individual call small and focused, which in practice produces more specific, less generic suggestions than a single request covering the entire goal at once. The resulting per-milestone subtopic lists are cached in the response payload so that the Execution agent can draw directly on them without re-querying the API.

3) Decision Agent: The Decision agent is the architectural centerpiece of the explainability claim made in this paper. It computes a milestone density value $\rho = m/d$, where m is the milestone count and d is the deadline in days, and applies a three-tier rule: ρ above 0.5 selects an Intensive strategy (three tasks per day, front-loaded), ρ between 0.2 and 0.5 selects a Balanced strategy (two tasks per day), and ρ at or below 0.2 selects a Relaxed strategy (one task per day). Because this logic is implemented directly in code rather than delegated to a language model, its output is fully deterministic: the same milestone count and deadline will always produce the same strategy, and the reasoning is stored verbatim in the audit log.

4) Execution Agent: The Execution agent takes the milestones, the per-milestone research subtopics, and the Decision agent's tasks-per-day limit, and materializes them into individual task rows in the plans table, each linked to its parent goal and milestone and tagged with a day number and a pending status. This agent performs no external API calls; it is a pure data-transformation step, which makes it the fastest stage in the pipeline and the least likely to fail independently of upstream agents.

5) Validation Agent: The Validation agent queries all tasks associated with a goal, computes the ratio of completed to total tasks, and classifies the goal's status as On Track (completion rate at or above 70%), Slightly Behind (40 to 69%), or Needs Replanning (below 40%). This agent is reinvoked every time a learner views the goal detail page or updates a task's status, so the displayed status is always current rather than a stale, one-time computation. Because it recomputes from the underlying task rows rather than caching a stored status field, the Validation agent cannot drift out of sync with the actual database state.

6) Memory Agent: The Memory agent persists learnerspecific key-value pairs — for example, the pacing strategy most recently chosen for that learner — in a dedicated table keyed by user ID. This is deliberately kept simple, a flexible key-value store rather than a fixed schema, so that future agents can persist new types of preference without a database migration.

7) Reporting Agent: The Reporting agent receives the goal title and the outputs of the Decision and Validation agents and asks the Gemini API to synthesize a short, encouraging, three-to-four sentence progress summary. As with the Planner and Research agents, a deterministic fallback sentence is generated locally if the API call is unavailable, ensuring the dashboard always displays a summary.

C. System Workflow

When a learner submits a new goal, the request reaches the Express backend, which authenticates the user via a JSON Web Token and invokes the orchestrator module. The orchestrator executes the seven agents strictly in sequence — Planner, Research, Decision, Execution, Validation, Memory, then Reporting — passing the output of each stage as input to the next, and writing every intermediate result to the agent_logs table. The final response returned to the React frontend includes the generated milestones, daily tasks, chosen strategy, current validation status, and the naturallanguage report, all rendered on the goal detail page alongside a collapsible view of the raw per-agent logs, which the authors use as a live demonstration artifact. Algorithm 1 summarizes the orchestration logic in pseudocode.

```
Algorithm 1: orchestrate(userId, goalId, title, desc, days)
1: milestones <- Planner(title, desc, days); log(Planner)
2: for each m in milestones do 3:     research[m] <-
Research(m, title) 4: end for; log(Research) 5: decision <-
Decision(len(milestones), days); log(Decision) 6: n <-
Execution(goalId, milestones, research, decision) 7:
log(Execution, n) 8: validation <- Validation(goalId);
log(Validation) 9:
Memory.remember(userId, decision.strategy) 10: log(Memory)
11: report <- Reporting(title, decision, validation) 12:
log(Reporting) 13: return {milestones, decision,
validation, report}
```

D. Implementation

The backend is organized into a dedicated agents directory containing one file per agent plus an orchestrator file that sequences their execution, keeping each agent independently testable and replaceable. Routes are separated into authentication and goal-lifecycle modules, with a shared middleware enforcing JWT verification on protected endpoints. The persistence layer uses SQLite via Node.js's

built-in node:sqlite module across five normalized tables, summarized in Table II, linked by foreign keys. This choice avoids the native-module compilation issues that alternative SQLite bindings can introduce on Windows-based development and demonstration machines, which was a practical constraint given that the system was developed and tested primarily on shared laboratory computers without administrative access for installing native build toolchains. To ensure the system remains operational even when the Gemini API is unreachable or misconfigured, a fallback layer in the client module returns clearly labeled mock output rather than allowing the pipeline to fail, so the full sevenagent chain, database writes, and user interface continue to function end-to-end regardless of external API availability.

E. Security and Privacy Considerations

User passwords are never stored in plaintext; each password is salted and hashed with bcrypt before being written to the users table, so that even direct database access does not reveal a learner's original credential. Session state is managed through stateless JSON Web Tokens rather than server-side sessions, which simplifies horizontal scaling and avoids the need for a shared session store. Every goal-related endpoint validates the token and scopes all database queries to the authenticated user's own records, preventing one learner from reading or modifying another learner's goals, tasks, or memory entries. Because the Decision agent is rulebased rather than LLM-driven, it introduces no risk of prompt injection influencing scheduling behavior; the LLM-driven agents (Planner, Research, Reporting) constrain the Gemini API to a narrow, structured task per call, which limits the surface area for unexpected model behavior relative to a single open-ended planning prompt.

F. Deployment Considerations

The current implementation targets single-machine, single-institution deployment, which matches the intended academic demonstration context: a lightweight Node.js process serving both the API and a locally built React bundle, backed by a single SQLite file. This keeps setup friction low — a genuine practical constraint on shared laboratory machines where administrative privileges and native build toolchains are often unavailable — since node:sqlite ships with the Node.js runtime itself and requires no separate database server or native compilation step. For a larger deployment serving multiple institutions concurrently, the same seven-agent boundaries defined in Section III-B would map naturally onto independently scalable services behind a shared API gateway, with SQLite replaced by a networked engine such as PostgreSQL; the orchestrator's sequential contract between agents would remain unchanged, since each agent already communicates through a well-defined input/output interface rather than shared mutable state.

TABLE II. DATABASE SCHEMA OVERVIEW

Table	Purpose
users	Account records with bcrypt-hashed passwords
goals	One row per learner-defined goal
plans	Individual daily tasks from the Execution agent
agent_logs	Full audit trail of every agent invocation
memory	Persisted learner preferences

IV. RESULTS AND DISCUSSION

A. Experimental Setup

The system was evaluated through functional and integration testing rather than a formal user study, since the primary contribution of this work is architectural rather than pedagogical. Backend endpoints for registration, login, goal creation, task-status update, and progress retrieval were verified individually to confirm correct request handling, database persistence, and JWT-protected access control. The full seven-agent pipeline was exercised end-to-end by creating goals with varying deadlines and milestone counts to confirm that the Decision agent selects different pacing strategies as milestone density changes, and that the Validation agent's status classification updates correctly as tasks are marked complete. The frontend was verified via a production build to confirm the absence of compilation errors. Table III summarizes the verification outcomes.

TABLE III. SYSTEM VERIFICATION SUMMARY

Test	Outcome
7-agent pipeline on goal creation	All 7 stages execute in order; logged
Decision strategy vs. density	Strategy changes consistently
Task completion -> Validation	Status recalculates correctly
Invalid Gemini API key	Fallback executes; no crash
Frontend production build	Completes with zero errors

A representative excerpt from the agent_logs table for a goal titled “Learn React Basics” with a 14-day deadline is shown below, illustrating the agent-level traceability described in Section III:

```
[Planner] decompose_goal -> 4 milestones,  day_number in {3,7,11,14} [Decision] choose_strategy -> {strategy: "Balanced", tasksPerDay: 2, reasoning: "milestone_density=0.29"} [Execution] create_tasks -> {insertedCount: 8} [Validation] check_progress -> {status: "Needs Replanning", completed: 0, total: 8}
```

B. Discussion

Because each agent's output is logged separately, the reasoning behind any generated plan can be inspected stage by stage, which distinguishes this architecture from singlemodel planning tools where such an audit trail is not available. The rule-based Decision agent, in particular, illustrates that not every stage of an agentic pipeline requires a large language model: selectively combining deterministic and LLM-driven components can improve explainability without sacrificing adaptability elsewhere in the pipeline. This observation echoes a broader theme in current agenticAI system design, where the choice of which sub-tasks to delegate to a language model versus conventional code is

itself a first-class design decision rather than an afterthought, and the Decision agent in this system was deliberately positioned as the boundary case that makes this trade-off visible to the reader.

The fault-tolerant design of the Gemini client also proved to be a practically significant result in its own right. During development, API misconfiguration did not halt testing of the orchestration logic, database schema, or frontend, because the fallback layer allowed the full pipeline to be exercised independently of external service availability. This separation of concerns between orchestration correctness and AI reasoning quality is a reusable pattern for other agentic systems that depend on third-party model APIs, and is particularly relevant for academic deployments where freetier API quotas or intermittent connectivity are common constraints.

A further observation from testing concerns the interaction between the Planner and Decision agents. Because milestone count is itself an output of an LLM-driven agent, small variations in Planner output (for example, five milestones instead of four for a similar goal) can shift the computed milestone density across a threshold boundary and change the selected strategy. This coupling is intentional — the Decision agent is meant to be sensitive to the Planner's actual output rather than to the raw goal text — but it does mean that Decision agent behavior is only as reproducible as the upstream Planner output for a fixed random seed, an important caveat when explaining the system's determinism claim precisely: the Decision function itself is deterministic given its inputs, even though one of its inputs originates from a non-deterministic language model call.

C. Threats to Validity and Limitations

A formal comparison of learning outcomes between GoalForge AI users and users of static planning tools has not yet been conducted and is identified as future work rather than a claimed result of this study. Similarly, the Decision agent's density thresholds of 0.2 and 0.5 were chosen heuristically based on reasonable pacing assumptions and have not been tuned against empirical learner outcome data; doing so is a natural next step once usage data becomes available. Agent quality for the Planner, Research, and Reporting stages depends on the underlying Gemini model and is subject to that model's general limitations, including occasional generic output for very broad or ambiguously worded goals. SQLite, while suitable for a single-institution academic deployment, would require migration to a networked database engine to support concurrent access at larger, multi-institution scale. Finally, the current evaluation was conducted by the authors during development rather than by independent testers, which is a common limitation of early-stage systems research and is mitigated here by the fact that the evaluated properties — correct sequencing, correct status transitions, and graceful degradation — are objective and independently verifiable from the logged data rather than subjective judgments.

V. CONCLUSION

This paper presented GoalForge AI, a seven-agent orchestration framework that reframes study planning as a coordinated pipeline of specialized, individually auditable agents rather than a single opaque model call. By combining LLM-driven agents for goal decomposition, research, and reporting with a deterministic, rule-based Decision agent, the system achieves a balance between adaptability and explainability that is difficult to obtain from a single end-toend model. The working implementation, spanning a Node.js and Express backend, an SQLite persistence layer, and a React frontend, demonstrates the technical feasibility of this architecture, and the persistent audit log provides a concrete mechanism for inspecting an agentic system's reasoning — a property of growing importance as such systems are adopted in education and other human-centered domains.

More broadly, the design pattern demonstrated here — pairing LLM-driven agents for open-ended reasoning tasks with deterministic, rule-based agents for safety- or trustcritical decisions — is not specific to study planning and may generalize to other agentic applications where explainability and predictable behavior are as important as adaptability. Future work will focus on four directions: empirical evaluation with real learners over a full academic term to measure plan adherence against a static-timetable baseline; an automatic replanning trigger so the Validation agent can invoke the Planner and Decision agents again when a learner falls significantly behind, rather than requiring manual goal recreation; richer memory modeling capable of capturing individual learning styles and historically effective task densities rather than a single last-used strategy; and migration of the persistence layer to a networked database engine to support concurrent, multi-institution deployment at scale.

ACKNOWLEDGMENT

The authors thank the PG Department of Computer Science and the Department of Information Technology at Syed Ammal Arts and Science College, Ramanathapuram, for the infrastructure and guidance provided during the development and testing of this system, and acknowledge the role of the seven-agent implementation in serving as a practical, handson vehicle for exploring agentic AI concepts within an academic project setting.

REFERENCES

[1]S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach," 4th ed., Pearson, 2021.

[2]T. Brown et al., "Language Models are Few-Shot Learners," in Proc. NeurIPS, 2020.

[3]Google AI, "Gemini API Documentation," Google DeepMind, 2025.

[4]OpenJS Foundation, "Node.js SQLite Built-in Module Documentation," Node.js Docs, 2025.

[5] React Core Team, "React Documentation," Meta Open Source, 2025.

[6]M. Wooldridge, "An Introduction to MultiAgent Systems," 2nd ed., Wiley, 2009.

[7] IEEE Computer Society, "IEEE Standard for Software and System Test Documentation," IEEE Std 829-2008, 2008.

[8] OWASP Foundation, "OWASP Authentication Cheat Sheet," OWASP Foundation, 2024.