Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Constraint-Safe Agentic Framework for Academic Task Planning Using ERP-Aware Capacity Modelling

Khushboo Kaushik

kaushik.khushboo09@gmail.com

The NorthCap University

Anushka Singh Dhillon

The NorthCap University

Nitin jakhar

The NorthCap University

Kunal godara

The NorthCap University

Monika yadav

The NorthCap University

Research Article

Keywords: Agentic AI, Academic Planning, Task Scheduling, ERP Integration, Plan Stability, LLM Advisor, Capacity Modeling

Posted Date: November 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11814388/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

A Constraint-Safe Agentic Framework for Academic Task Planning Using ERP-Aware Capacity Modelling

Khushboo Kaushik*
Dept. of Computer Science and Engineering
The NorthCap University, Gurugram, India
kaushik.khushboo09@gmail.com

Anushka Singh Dhillon
Dept. of Computer Science and Engineering
The NorthCap University, Gurugram, India
anushkadhillon369@gmail.com

Nitin Jakhar
Dept. of Computer Science and Engineering
The NorthCap University, Gurugram, India
nitinjakhar5703@gmail.com

Kunal Godara
Dept. of Computer Science and Engineering
The NorthCap University, Gurugram, India
Kunalgodara19@gmail.com

Monika Yadav
Dept. of Computer Science and Engineering
The NorthCap University, Gurugram, India
Yadavmonika506@gmail.com

Abstract—Students miss deadlines because the information about tasks is spread out across email, the learning management system (LMS) and the institutional ERP and there is no tool that checks the information against the time a student actually has. This paper introduces CollegeOS, an agentic planning framework that senses academic tasks, ranks them with a four-signal priority score, packs a day-by-day study schedule into a capacity model reduced by the student's real class timetable, and repairs the plan when conditions change. The framework is by construction constraint-safe: a deterministic scheduler enforces deadlines and daily capacity as hard constraints, while an optional large language model (LLM) can only provide bounded priority boosts and can never change feasibility. The connector for the NorthCap University ERP provides the attendance and timetables, which feed the capacity model and attendance-aware boosts. We evaluate the system against an earliest-deadline-first (EDF) baseline over 40 simulated semesters. Repair-mode replanning reduces mean plan churn from 0.899 to 0.813 (the 95% confidence intervals do not overlap), while on-time completion (50.2% vs. 51.0%) is indistinguishable from the baseline in the saturated workload regime tested. We report these results openly, along with a weight ablation and the known limitations, and identify a feasibility-aware scheduler as the next step.

Index Terms— Agentic AI, Academic Planning, Task Scheduling, ERP Integration, Plan Stability, LLM Advisor, Capacity Modeling.

I.Introduction

Academic life has moved online bit by bit. Assignments and grades are in learning management systems (LMS) such as Canvas [1], announcements arrive by e-mail [2], attendance and records reside in the institutional enterprise resource planning (ERP) portal, and the resulting schedule typically resides in a calendar that the student maintains by hand. Each system is the authority for its own slice of the student's workload, but none of them looks at the others. The student has to cross-check deadlines, guess how long each task will take, and rebuild the plan every time something changes. Deadlines are missed not because the student is not trying but because there is no check on the list of things to do versus the time the student actually has. We see this as an agent problem with well defined steps: sense, prioritize, plan, monitor and replan. Recent work on large language model (LLM) agents argues that the model should not be the sole source of hard constraints [5]. We take that advice very seriously. In CollegeOS the planning core is deterministic, i.e., deadlines and daily capacity are hard constraints that cannot be overridden by any other component, and the LLM is reduced to an optional advisor that can only nudge priorities. Capacity is not a flat number, it is calculated from the real class timetable of the student taken from the university ERP, so a packed class day means less study time than a light one.

The paper makes the following contributions:

1) ERP-aware capacity model that reduces the study budget for each day by the timetabled class minutes and attendance-based priority increases from ERP attendance records.
2) A constraint-safe architecture with a deterministic scheduler enforcing capacity and due dates, an optional LLM advisor limited to bounded boosts, and all external writes propose-then-approve.
3) A replanner operating in repair mode and a plan-churn metric measuring the extent to which a replan disrupts the student's current plan.
4) Open simulation study on 40 semesters vs. an EDF baseline, with a decomposition and a weight ablation, reporting negative as well as positive results.

The remainder of the paper is organized as follows: Section II describes related work. Section III defines the problem. Section IV describes the proposed approach. Section V discusses the experimental setup. Section VI discusses the results. Section VII compares CollegeOS with existing tools. Section VIII describes the limitations. Section IX concludes.

II. Related Work

A. LLM Agents and Planning

Huang et al. [3] provide a review of LLM-based agent planning and categorize the current approaches into task decomposition, plan selection, external modules, reflection, and memory. The external-module approach is closest to our design, where the model is assisted by other components, rather than planning only. ReAct interleaves reasoning traces with actions so that the model can form, track and update plans and handle exceptions [4]; however, the model itself decides what to do next, so guaranties on constraints such as deadlines must come from

elsewhere. Kambhampati et al. claim that auto-regressive LLMs cannot plan and self-verify on their own, and propose LLM-Modulo frameworks, where the model is combined with external model-based verifiers [5]. We run a deterministic scheduler and an advisory LLM to support this view. However, none of these works address personal scheduling against a fixed daily time budget.

B. Deadline-Based Scheduling

Earliest-deadline-first (EDF) was analyzed by Liu and Layland for hard real-time systems [6]. It always runs the task with the nearest deadline. It ignores both task importance and work left, which makes it a natural baseline: in CollegeOS it is the same prioritizer with the urgency weight set to one and all other weights set to zero.

C. Student-Facing Tools

Motion [7] is an AI calendar that automatically schedules tasks and replans when things change. Motion is not designed for academic life, has no connection with attendance and, to our knowledge, publishes no evaluation of its plans. Notion [8] is a flexible workspace in which everything is entered and updated manually. It does not prioritize, schedule or replan. Canvas To-Do [1], and Coursicle [9] are student-oriented and suck in some deadlines automatically, but they list what's due rather than rank tasks, build a schedule, or adjust it when work falls behind.

D. Research Gap

Research on LLM planning shows that hard constraints should not be left entirely to the model. This idea has not yet been used in student scheduling. Current scheduling tools do not include academic details, like attendance records or class timetables. Also none of the tools we looked at share any evaluation or explain how much their plans change when they are replanned. CollegeOS is built to fix this gap.

III. Problem Definition

Let T be the set of tasks. Task i has a date D_i, a believed remaining effort R_i in minutes and a declared importance $I_i \in \{1, \dots, 5\}$. Let $C(d)$ denote the study minutes on day d. A plan assigns $x_{i,d} \geq 0$ minutes of task i to day d. Is feasible if

$$\sum_i x_{i,d} \leq C(d) \quad \text{for all} \quad d \tag{1}$$

$$x_{i,d} = 0 \quad \text{for} \quad d > D_i, \quad \sum_d x_{i,d} \leq R_i \tag{2}$$

Work that cannot be placed under these constraints is reported as at risk rather than forced into the plan. The challenge is to keep a plan even when new tasks come effort numbers change and classes move. The plan should (a) put much remaining effort as possible before each deadline and (b) stay close, to the previous plan because big changes make the student doubt the schedule.

The data that defines C(d) must be taken directly from the institution's systems. It should not be entered by hand again. Also no data should be sent to any system unless the student gives permission.

IV. Proposed Approach

A. Architecture

Fig. 1 Shows the architecture.

CollegeOS works as a loop that uses a long-lasting state object.Surrounding CollegeOS is a connector layer that keeps all dependencies at a safe distance.One cycle contains five stages. Sense: connectors pull tasks merge them into the state and remove duplicates; attendance and timetable data, from the ERP go straight into the state because they are not shaped as tasks.Prioritize: the monitor derives.

Three main ideas guide the design. First the planning core is deterministic. Does not need network access so it can be tested offline. Second all outside access goes through a connector interface that uses propose-then-approve writes. Third the LLM is advisory only.

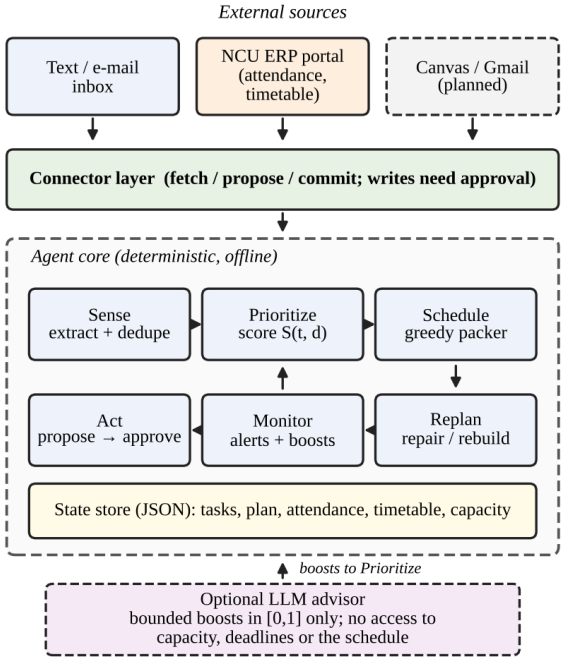

Fig. 1. CollegeOS architecture. The deterministic agent core is surrounded by a connector layer; the optional LLM advisor influences only priority boosts.

B. ERP-Aware Capacity Model

The capacity model tells us how time a student has available for studying on any particular day. The base budget B_d is 180 minutes on weekdays and 300 minutes on weekends. When the schedule is connected from the ERP system the class time K_d for day d takes away from this budget.

$$C(d) = \min(B_d, \max(0, T - K_d)) \tag{3}$$

where T equals 480 minutes which's the total academic day including classes and study time. A day with classes keeps the base budget the same. A day with classes reduces the base budget. The total capacity $C_{cum}(d_1, d_2)$ which is the sum of $C(d)$ from day d_1, to day d_2 is used by the pressure signal that comes next.

C. Prioritizer

For task i on day d the prioritizer gives a sum of four signals:

$$S = 0.35 \cdot u + 0.20 \cdot w + 0.40 \cdot p + 0.05 \cdot b \quad (4)$$

$$u = \frac{1}{1+\delta}, \text{ where } \delta = D_i - d \quad (5)$$

$$w = \frac{I_i - 1}{4} \quad (6)$$

$$p = \min\left(1.5, \frac{R_i}{c_{\text{cum}}(d, D_i)}\right) \quad (7)$$

With $p = 1.0$ when no capacity remains. Urgency u increases quickly as the deadline gets closer. Weight w scales the declared importance to a value between 0 and 1. Pressure p measures the work left compared to the time left. This is what sets the policy apart from a simple sorted to-do list. Boost b is a value between 0 and 1 that comes from attendance or advisor input, for each course. The weights are given as constructor arguments. Are easy to adjust or remove.

D. Scheduler

The scheduler works by packing tasks one day at a time over a 21-day period following Algorithm 1. Each day it looks at all tasks that can be done and ranks them based on a prioritizer score. This score is calculated using a snapshot of the state, where the remaining effort already takes into account the blocks that have been scheduled so far. The ranked task gets a block of time.

Block sizes are limited. The smallest block is 30 minutes. Smaller blocks are only used if they finish a task completely. No block can be than 90 minutes in a single sitting. Also no task can get than 180 minutes of time in a single day. This ensures that no one task dominates the day and that switching between tasks still makes sense in terms of time cost.

After assigning blocks any blocks for the task on the same day are combined into one. If a task still has work left that hasn't been scheduled it is added to an, at-risk map. This map tracks how many minutes of work are still missing for that task.

Algorithm 1: Constraint-Safe Greedy Scheduling

Input: open tasks T, capacity $C(d)$, horizon of 21 days, pinned blocks P (empty for rebuild)
Output: plan, at-risk map A

1. Place the blocks in P first, bounded by remaining effort and free capacity
2. **for** each day d in the horizon **do**
3. $f \leftarrow C(d)$ − minutes already placed on d
4. **while** $f \geq 30$ **do**
5. $E \leftarrow \{i : R_i > 0, D_i \geq d, \text{placed}(i, d) < 180\}$
6. **if** $E = \emptyset$ **then break**
7. $k \leftarrow \arg\max_{i \in E} S(i, d)$, using current remaining effort
8. $s \leftarrow \min(90, R_k, f, 180 - \text{placed}(k, d))$
9. place s min of k on d (below 30 min only if $s = R_k$)
10. update R_k, placed(k, d) and f
11. merge same-day blocks of the same task
12. **for** each task i with $R_i > 0$ **do** $A[i] \leftarrow R_i$

Property 1 (feasibility). Every plan created by Algorithm 1 meets conditions (1) and (2) no matter what boost signals are used. The system checks capacity and due dates when it places blocks (lines 4–9). The score S is only used to rank candidates so nothing that goes into the score—including a boost from an LLM—can make a block go beyond capacity or after its deadline. Problems appear as at-risk entries. That comes from the placement rules. We have not proven it with a separate formal proof.

E.. Attendance Boosts. The monitor raises four kinds of alerts: critical) for open tasks past their due date; overrun (warning) when logged effort exceeds the estimate by more than 25%; at-risk (critical) for each entry, in the shortfall map; and attendance (warning).

when a course is below its threshold θ, which is set at 0.75 by default. The ratio is $r = \frac{a}{h}$ for a course with h classes held and a attended. The number of subsequent classes required to recover is

$$n^* = \left\lceil \frac{\theta h - a}{1 - \theta} \right\rceil \quad (8)$$

In order to provide a troubled course extra study attention, the same gap results in a scheduling boost for the course,

$$b_c = \min\left(1, \frac{\theta - r}{\theta}\right), \text{ for } r < \theta \quad (9)$$

F. Replanner and Plan Churn

To be able to directly compare two replanning strategies, two replanning strategies are implemented. Rebuilding throws away the existing schedule and builds a brand new one from scratch that matches the new state, but it might reshuffle everything the student had planned. Repair pins every block that is not finished and whose day is between today and the deadline. Only the rest of the work is scheduled around the pinned blocks. We measure disruption in terms of plan churn:

$$\text{Churn} = \frac{|\{(i,d) \in K : m_{\text{old}}(i,d) \neq m_{\text{new}}(i,d)\}|}{|K|} \quad (10)$$

where $m_{\text{old}}(i, d)$ and $m_{\text{new}}(i, d)$ are the number of minutes allocated to task i on day d in the old and the new plan, and K is the set of keys (i, d) that appear in either plan. 0 means the new plan is the same as the old plan 1 means every assignment changed.

G. Extraction and LLM Advisor

Tasks enter through two interchangeable pullers. The rule-based extractor breaks text into chunks, retains those containing action words such as due, deadline, submit, exam or quiz, resolves dates by an ordered list of patterns (ISO dates, day-first numeric dates, "4 October", "October 4th" and relative expressions such as tomorrow or next Friday) and infers a task type from keyword hints, filling in default effort and importance by type (for example assignment 180 min, quiz 60 min, exam 480 min). The interface is the same as the LLM extractor, it expects a JSON array, validates each field, and falls back to the rule extractor on any failure. Both outputs are de-duplicated such that two tasks are duplicates if they share a course and date due and the Jaccard overlap of their title words is greater than 0.5.

All model calls are made through a generic helper which will choose between Anthropic [11] or Gemini [10] based on key set in config. If there is a failure, this will return empty and we can always fall back onto the deterministic approach. The advisor gets a list of open tasks, active alerts, and whether the plan is feasible. It returns [0,1] boosts, a brief 2-3 sentence reason for the boost to the student, and a list of suspicious signals. Boost keys are checked to be in the list of open task ids. Values are clamped, and text is truncated. Attendance and advisor boosts are summed, then clamped to 1.0. Due to Property 1 incorrect

or missing advisor responses can only change focus but won't create an infeasible plan.

H. ERP Connector and Privacy

The ERP connector uses a bearer token from the student's current browser session to access the JSON API of the portal. There is no automatic login or storing of any credentials. By inspecting the portal we found two endpoints:

The network communications layer contains the attendance summary and timetable. This is received within an outer envelope containing data in the form of a JSON string. This means that validation can happen on every level. If there is no token, token is expired (HTTP 401/403), error or unexpected response then errors are raised explicitly. The faculty names and images have been removed intentionally as all the planner requires to know is when the student is busy. Currently the connector is pointing to a staging host (UAT) with undocumented endpoints. It is therefore required to confirm long term availability with IT at the institution (see Section VIII). All data is stored locally in a JSON file. The framework has been developed in Python using only the standard library

V. Experimental Setup

We do not have sufficient real data at this stage to fairly compare policies. We therefore evaluate the planner as a scheduler within a built-in simulation framework. The following scenario is simulated: A 45 day long semester starting on 3rd August 2026 containing 22 tasks across 5 different courses.

Task Generation

Each task was generated with task types weighted by 5:3:1:3:2:2 for Assignment, Quiz, Exam, Reading, Project and Admin. Each task is generated on a random day with a lead time between 4-18 days before the due date. The planner observes a noisy estimate of the task. This is calculated by multiplying the default time of the task type by a random uniform variable between 0.6-1.4. The actual effort required was derived by multiplying this by a uniformly distributed variable between 0.7-2.0 (Minimum 20min). This means that on average the actual effort is around 1.35 times larger than the estimated value. The student can only discover this by completing the task.

Student Behaviour

The student will complete each scheduled block with some imperfection each day. The number of minutes completed will be a random variable between 0.6-1.05 times the block size. However, the student will not complete more than the available capacity for the day. If the student completes the estimated number of minutes and still hasn't completed the task then there has been an overrun. The estimated value for the task will be increased by a factor of 1.35 and 1.10 for the agent and baseline respectively.

Policies: The EDF policy uses urgency weight of 1.0 and all other weights set to zero. The agent uses default weights of (0.35, 0.20, 0.40, 0.05) with re-plan in repair mode and re-estimation factor of 1.35.

Performance Measures: % on time, total lateness in days, average number of changes made to the plan during the semester, and crunch days (days when >90% of capacity is allocated). Both policies used seed values between 0 and 39 meaning that the same set of tasks will be used in both sets of runs. 95% confidence intervals were found using the normal approximation of +/- 1.96 standard error.

VI. Results and Discussion

A. Agent vs. EDF Baseline

These results are summarised in Table I and shown graphically in Fig. 2. It can clearly be seen that there is a significant reduction in plan churn by using the repair mode replanning approach. The average plan churn has decreased by 0.086 (from 0.899 to 0.813). However, the confidence intervals do not overlap. There is also little difference between the two policies in terms of deadline performance. The confidence intervals are much larger than the difference in on time completion (+0.8 points), lateness (-0.7 days) and crunch days (-0.2).

TABLE I
AGENT VERSUS EDF BASELINE (MEAN ± 95% CI, 40 RUNS)

Metric	EDF	Agent	Delta
On-time completion (%)	50.2 ± 3.7	51.0 ± 4.1	+0.8 (n.s.)
Total lateness (days)	169.7 ± 14.6	169.0 ± 15.0	−0.7 (n.s.)
Mean plan churn	0.899 ± 0.016	0.813 ± 0.021	−0.086
Crunch days	12.9 ± 1.6	12.7 ± 1.4	−0.2 (n.s.)

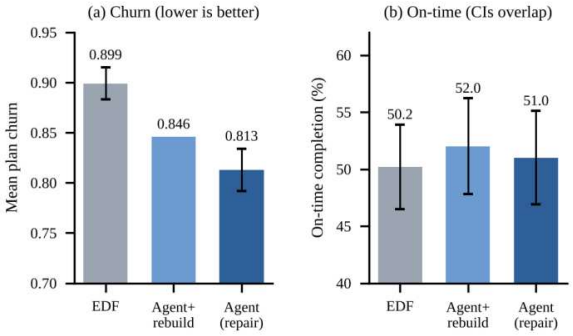

Fig. 2. Plan churn and on-time completion (mean ± 95% CI over 40 runs). No CI is available for the rebuild configuration's churn, and the churn axis is truncated.

B. Isolating the Effect of Repair

As the agent has three differences compared to the baseline, we ran an additional configuration using the agent's prioritizer and re-estimator with the rebuild mode planner (Table II). We can see that approximately half (0.899 to 0.846) of the decreased churn is caused by the prioritizer and re-estimator of the agent. The other half (0.846 to 0.813: around 3.3 percentage point difference) is due to repair. So we can conclude that repair is helpful but does not contribute as much as the figures suggest. Both changes have little impact on meeting the deadlines.

TABLE II
DECOMPOSITION OF THE CHURN REDUCTION

Configuration	On-time (%)	Late (d)	Churn
EDF baseline	50.2 ± 3.7	169.7	0.899
Agent priorities + rebuild	52.0 ± 4.2	163.6	0.846
Agent (repair)	51.0 ± 4.1	169.0	0.813

C. Prioritizer Weight Ablation

Table III shows a variation of weights used by the prioritizer. All variations produce an on-time completion rate within the confidence intervals of each other. Removing the pressure term has no negative effects on metrics. However, we had expected based on the design that removing pressure would cause a regression to EDF, but this is not supported by the results of this simulation. No weighting performs better than another so we cannot claim that the default weights are fine-tuned.

TABLE III
WEIGHT ABLATION (URGENCY / IMPORTANCE / PRESSURE / BOOST)

Weights	On-time (%)	Late (d)	Churn
Default (.35/.20/.40/.05)	51.0 ± 4.1	169.0	0.813
No pressure (.35/.20/0/.05)	52.0 ± 3.6	165.8	0.814
No urgency (0/.20/.40/.05)	51.0 ± 4.2	172.7	0.797
No importance (.35/0/.40/.05)	52.2 ± 3.5	167.3	0.818
Pressure only (0/0/1.0/0)	50.1 ± 3.8	168.3	0.794
Pressure-heavy (.2/.1/.7/0)	52.0 ± 3.7	166.8	0.805

D. Interpretation and Threats to Validity

Saturated Regime: All policies perform roughly half of the tasks within deadline. The true effort is approximately 1.35 times that of the estimate and therefore the average compliance is ~0.83. This means the effective demand felt by the student is significantly higher than planned, however the total weekday time window is fixed at 180 minutes. Due to the system being so tight, the order of servicing tasks makes very little difference leading to the similar policies. Lower and variable demand scenarios are needed to see if pressure aware scheduling has an impact in the presence of slack.

Edge Effect: Around 8% of the tasks that were generated had deadlines beyond the 45 day planning horizon. Removing these results in 50.4% (EDF) and 51.5% (agent). This leads us to the same conclusion. Confounding Factors: The priorities, replanning mode and re-estimation factor all differ between the agent and baseline. As Table II only differentiates between repair and the other two factors, a full factorial is required. Synthetic Assumptions: Whilst the noise in effort, compliance and arrivals are based on our assumption.

E. Live Case Study

The end-to-end functionality was tested using a state snapshot captured on 28 September 2026. This included 7 tasks, 6 ERP attendances, 6 classes in the timetable over 2 days and 10 blocks in the plan. There are 3 tasks to complete, all of which have been scheduled. Each day there are enough tasks scheduled to fill the available 180 minutes each weekday, between 28 September and 2 October. On Saturday 3 October there are 120 minutes being utilised out of 300 minutes available. This means that the plan is full but achievable with no open items to risk. All courses have 75%+ attendance with CSL448 having the lowest attendance at 78.6%. There are 160 minutes of classes in the timetable on 29 September and 200 minutes on 1 October. However, with 480 minutes per day there is still more than 180 minutes available on both days. Two issues were highlighted by this snapshot and can be seen in Section VIII. The first issue is that there are 4 tasks overdue reported twice with no study blocks allocated. The second issue is that the course codes used in the ERP do not match the course codes used in the tasks (e.g. CSL448 vs CS401). This means that no extra attendance could be applied.

VII. COMPARISON WITH EXISTING TOOLS

Table IV shows a comparison between CollegeOS and the tools discussed in section II. LMS sensing is still going to be implemented, so the sensing capability is considered partial. From section VI, it can be seen that we cannot claim any advantage over a baseline scheduler, since it only demonstrates a reduced planning variability.

TABLE IV
CAPABILITY COMPARISON

Capability	Motion	Notion	Canvas	Course.	CollegeOS
Automated sensing	No	No	Partial	Partial	Partial
Priority beyond deadline order	Yes	No	No	No	Yes
Deadline-aware auto-scheduling	Yes	No	No	No	Yes
Automatic replanning	Yes	No	No	No	Yes
Attendance-linked monitoring	No	No	No	No	Yes
Academic-context specific	No	No	Yes	Yes	Yes
Evaluated against a baseline	Not publ.	N/A	N/A	N/A	Yes

VIII. LIMITATIONS

Here we outline the main limitations of our code during the code review and live snapshot. (1) We cannot schedule past due tasks as the scheduler requires due_date >= day. These will show up as both overdue and at risk. (2) Our greedy allocation does not consider feasibility. The scoring method may allocate capacity to a task which cannot be used in later deadlines. To overcome this we will use a backwards pass. (3) The pressure term is unproven. We cannot differentiate between weightings in the ablation. (4) ERP codes and task course codes do not match. So student attendance never triggers. We need to map these. (5) ERP dependency. We are relying on endpoints that are undocumented, hosted on a staging host and require a token that has been manually copied and will expire. It would be good to have a fallback of importing from CSV. (6) Capacity is only based on timetabled classes. It does not account for sleeping, travelling or personal time. (7) The factors (1.3, 1.35, 1.10) are not fitted. (8) The extraction accuracy and recall has not been assessed. We have not tested how the advisor affects the quality of the plan. The only thing we know is that it is safe (Property 1).

IX. CONCLUSION AND FUTURE WORK

This paper introduces CollegeOS, a constraint-safe agentic system that builds and maintains a daily academic schedule based on collected tasks. The key features are calculating capacity using the student's true time-table and having hard constraints managed by a deterministic scheduler, whilst only allowing the LLM to give limited advice. Repair mode replanning within the agent reduced the mean number of changes made to the plan during simulated scheduling (0.899) to (0.813) compared to an EDF baseline. Of this reduction around 0.033 can be attributed to the repair functionality. The on-time completion, lateness and crunch days showed no

There was no measurable difference. Weight ablation also failed to justify using the pressure term in this regime. These findings are useful to know for the next steps.

(i) Backprop scheduler that will take into account feasibility. (ii) Defects fixed relating to late tasks and mapping to course codes. (iii) Lighter and more variable regimes. Factorial design with prioritisation, replanning mode and re-estimation. (iv) Live integration with Gmail and Canvas. Write to these systems only when approved. Fallback to CSV for ERP. (v) Set of around 200 real messages with labels. Report precision and recall for both extractors. Unit tests and assess how much the advisor improves the plan quality.

References

1) Ahmetoglu, Y., Brumby, D. P., & Cox, A. L. (2024). Bridging the gap between time management research and task management app design: A study on the integration of planning fallacy mitigation strategies. In Annual Symposium on Human-Computer Interaction for Work (CHIWORK) (pp. 1–14).
2) Ahmetoglu, Y., Iskandar, A., Taoh, S., Ying, A., Brumby, D. P., & Cox, A. L. (2025). "I work much better by doing less": How task duration feedback affects optimistic planning bias in academic work. In Annual Symposium on Human-Computer Interaction for Work (CHIWORK) (pp. 1–17).
3) Buchholz, M., Heins, M., & Schmidt, M. (2026). The impact of rescheduling on production stability. Zeitschrift für wirtschaftlichen Fabrikbetrieb, 121, 35–40.
4) Buzzega, G., & Montangero, M. (2024). Characterizing global work-conserving scheduling tardiness with uniform instances on multiprocessors. Real-Time Systems, 60, 537–569.
5) Buzzega, G., Nocetti, G., & Montangero, M. (2023). Characterizing G-EDF scheduling tardiness with uniform instances on multiprocessors. In International Conference on Real-Time Networks and Systems (RTNS) (pp. 45–55).
6) Chang, E. Y., & Geng, L. (2025). SagaLLM: Context management, validation, and transaction guarantees for multi-agent LLM planning. Proceedings of the VLDB Endowment, 18, 4874–4886.
7) Gundawar, A., Valmeekam, K., Verma, M., & Kambhampati, S. (2024). Robust planning with compound LLM architectures: An LLM-Modulo approach. arXiv.
8) Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., & Murthy, A. (2024). LLMs can't plan, but can help planning in LLM-Modulo frameworks. arXiv.
9) Kulcsár, G., Forrai, M. K., & Cservenák, Á. (2025). Objective functions for minimizing rescheduling changes in production control. Automation, 6, Article 30.
10) Li, Z., Hua, W., Wang, H., Zhu, H., & Zhang, Y. (2024). Formal-LLM: Integrating formal language and natural language for controllable LLM-based agents. arXiv.
11) Mejía, G., Montoya, C., Bolívar, S., & Rossit, D. A. (2022). Job shop rescheduling with rework and reconditioning in Industry 4.0: An event-driven approach. The International Journal of Advanced Manufacturing Technology, 119, 3729–3745.
12) Parmar, H. K., & Raman, S. (2026). Smart manufacturing scheduling under data latency: A rolling-horizon two-stage MILP framework for OEM–Tier-1 coordination. Journal of Manufacturing and Materials Processing, 10(4), Article 142.
13) Patel, V. (2026). Studyflow: The smart study planner. International Journal of Science Strategic Management and Technology, 2(5), 1–9.
14) Singh, J., & Singh, S. P. (2012). Schedulability test for soft real-time systems under multiprocessor environment by using an earliest deadline first scheduling algorithm. arXiv.
15) Stoyenko, A. D., & Georgiadis, L. (1992). On optimal lateness and tardiness scheduling in real-time systems. Computing, 47, 215–234.
16) Tighazoui, A., Sauvey, C., & Sauer, N. (2021). Predictive-reactive strategy for identical parallel machine rescheduling. Computers & Operations Research, 134, Article 105372.
17) Wang, C.-C., & Chiou, W. (2020). Greater required effort may induce closer perceived proximity to the task deadline, leading to less underestimation of task completion time. Educational Psychology, 42, 108–118.
18) Yang, Z., Raman, S. S., Shah, A., & Tellex, S. (2024). Plug in the safety chip: Enforcing constraints for LLM-driven robot agents. In IEEE International Conference on Robotics and Automation (ICRA) (pp. 14435–14442).
19) Zhao, S., Sahebi, S., & Feyzi-Behnagh, R. (2023). Curb your procrastination: A study of academic procrastination behaviors vs. a planning and time management app. In ACM Conference on User Modeling, Adaptation and Personalization (UMAP) (pp. 124–134).
20) Li, X., Choe, K. W., Liu, Y., Chen, X., Tao, C., You, B., Chen, W., Di, Z., Sun, J., Zheng, S., Bao, J., Wang, Y., Yan, W., Li, Y., & Lee, H.-C. (2026). ClawsBench: Evaluating Capability and Safety of LLM Productivity Agents in Simulated Workspaces. arXiv (Cornell University), abs/2604.05172.
21) Wijerathne, O., Nimasha, A., Dushan, F., De Silva, N., & Perera, S. (2025). ScheduleMe: Multi-Agent Calendar Assistant. arXiv (Cornell University), abs/2509.25693.
22) Wang, H., Poskitt, C. M., & Sun, J. (2026). AgentSpec: Customizable Runtime Enforcement for Safe and Reliable LLM Agents. ACM/IEEE International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2938-2950.
23) Miculicich, L., Parmar, M., Palangi, H., Dvijotham, K., Montanari, M., Pfister, T., & Le, L. T. (2025). VeriGuard: Enhancing LLM Agent Safety via Verified Code Generation. arXiv (Cornell University), abs/2510.05156.
24) Turan, E. (2026). Oversight Has a Capacity: Calibrating Agent Guards to a Subjective, Fatiguing Human. arXiv (Cornell University), abs/2606.08919.
25) Turan, E. (2026). Oversight Has a Capacity: Calibrating Agent Guards to a Subjective, Fatiguing Human. arXiv (Cornell University), abs/2606.08919.
26) M., Al-Shamayleh, A. S., Hourani, M., Mustafa, G., Khalid, Y. N., & Mumtaz, M. (2026). DBS-W: A Resource- and Deadline-Aware Scheduler for DAG-Structured Cloud Workflows. Applied Computational Intelligence and Soft Computing, 2026.