AI Agent–based Component Rescheduling Framework for Industrialized Construction

Huayu Zhong[1], Hui Lu[2], Ke Chen[1]

[1]School of Civil and Hydraulic Engineering, Huazhong University of Science and Technology, Wuhan, China
[2]School of Economics and Management, China University of Geosciences, Wuhan, China
hyzhong@hust.edu.cn, luhui@cug.edu.cn, chenkecm@hust.edu.cn

Abstract –

Industrialized construction relies on the tight integration of production, logistics, and on-site assembly, making the process highly sensitive to dynamic disturbances. To achieve rescheduling with minimum human intervention, this study proposes an intelligent rescheduling framework based on AI agents. First, we establish a unified Mixed-Integer Linear Programming (MILP) model to provide a standardized mathematical foundation for the rescheduling process. Second, a collaborative architecture comprising five functional AI agents is developed. By utilizing Large Language Models (LLMs) as their cognitive cores, these agents automatically transform natural-language disturbance reports into structured model patches and executable code. To ensure technical precision, a Retrieval-Augmented Generation (RAG) mechanism is integrated to ground the agents in domain-specific knowledge. Experimental results demonstrate that the framework can effectively bridge the gap between unstructured field reporting and mathematical optimization, enabling automated responses to disruptions.

Keywords –

Industrialized Construction; Dynamic Rescheduling; AI Agents; Framework

1 Introduction

Industrialized construction shifts traditional cast-in-situ construction methods toward a highly coordinated process of component production, transportation, and assembly. This transformation imposes spatiotemporal synchronization constraints across factories, logistics, and on-site resources[1]. Guided by just-in-time principles, buffer resources throughout the supply chain should be minimized, rendering the supply chain highly sensitive to various disturbances[2]. Consequently, local delays can propagate through the chain[3], triggering cascading effects such as vehicle idling[4] and on-site work stoppages[5]. Without mechanisms to generate actionable instructions, these chain reactions lead to overall performance degradation[6]. Therefore, prefabricated component scheduling requires not only robust static optimization but also agile operational rescheduling to mitigate disruptions.

Existing research in this domain spans from centralized optimization to distributed collaboration. While centralized approaches pursue global optimality, they often suffer from computational latency and lack the flexibility required to handle frequent disturbances[7-8]. To overcome these limitations, research has increasingly adopted Multi-Agent Systems (MAS) [9]. By modeling resources or tasks as autonomous entities with independent decision-making capabilities, MAS offer an adaptive alternative rooted in collaboration[10]. However, real-world disturbances are typically communicated via unstructured natural language such as site logs and verbal change orders. Traditional MAS, restricted by hard-coded decision logic[11], lack the ability to interpret ambiguous information and autonomously construct mathematical models. As a result, the pipeline from disturbance perception to decision generation remains heavily reliant on human intervention.

The emergence of Large Language Models (LLMs) provides a transformative pathway to overcome these hurdles. The primary advantage of LLMs lies not in replacing specialized optimization solvers, but in their capacity to serve as semantic translators[12]. This capability has catalyzed the rise of AI agents—autonomous entities that utilize LLMs as a cognitive core to perceive, reason, and act. An AI agent does not merely follow rigid rules; it interprets the intent behind a disturbance and manipulates computational tools to synthesize solutions.

Accordingly, this study proposes an intelligent rescheduling framework driven by AI agents. We first establish a unified Mixed-Integer Linear Programming (MILP) model to provide a standardized, solvable mathematical foundation for rescheduling[13]. Building on this, the framework employs AI agents as semantic-to-code bridges. By incorporating domain-specific

constraints through Retrieval-Augmented Generation (RAG), these agents automatically map unstructured disturbances into structured model modifications and executable solver code[14]. A collaborative workflow among specialized agents helps to achieve a fully automated loop from semantic disruption reporting to executable rescheduling commands.

2 Problem Description and Model Construction

This study specifically addresses the dynamic rescheduling of precast concrete components under disturbances. Grounded in the concept of interrupt management, the rescheduling strategy aims to minimize deviations between the revised schedule and the original baseline plan[15]. The optimization objective functions as a multi-criteria priority, balancing factors including (1) reducing costs associated with early or delayed component arrivals and (2) minimizing temporal shifts and resource reallocation overhead caused by schedule adjustments.

Coordination between the production facility and the construction site is maintained through the dynamic balancing of on-site inventory levels I_t. This framework adopts an event-driven rescheduling mechanism: upon the detection of a disturbance, a specialized Patch Agent translates the unstructured information into localized updates to automatically generate a reschedule solution.

Based on the problem description, the collaborative scheduling model is constructed as follows.

$$min\ Z = c_{late} * \sum_i l_i + c_{early} * \sum_t e_i + k_{time} * \sum_i shift_i^{time} + k_{line} * \sum_i \varphi_i^{line} \quad (1)$$

Equation (1) formulates the minimization objective, where c_{late} and c_{early} represent the unit penalty costs for the late arrival period l_i and early arrival period e_i of component i, respectively. To ensure schedule stability, the objective also penalizes deviations: k_{time} and k_{line} denote the penalty coefficients for temporal shifts and production line reallocations. Here, $shift_i^{time}$ represents the absolute deviation of the casting start time relative to the baseline, and φ_i^{line} is a binary variable indicating whether the production line assignment for component i has changed.

$$C_{i,o} = S_{i,o} + p_{i,o}, \forall\, i \in I \quad (2)$$

$$S_{i,od} \geq C_{i,oc} + \theta_i, \forall\, i \in I \quad (3)$$

$$S_{j,oc} \geq C_{i,od} - M_{big} * (1 - u_{i,j,l}^{line}) - M_{big} * (2 - x_{i,l} - x_{j,l}), \forall i, j \in I, i \neq j, \forall\, l \in L \quad (4)$$

$$S_{i,oc} \geq C_{j,od} - M_{big} * u_{i,j,l}^{line} - M_{big} * (2 - x_{i,l} - x_{j,l}), \forall i, j \in I, i \neq j, \forall\, l \in L \quad (5)$$

Equations (2-5) define the physical production logic. In Equation (2), $S_{i,o}$ and $C_{i,o}$ denote the start and completion times of process o (containing casting o_c and demolding o_d), and $p_{i,o}$ represents the processing duration. Equation (3) strictly enforces the curing constraint, requiring that the demolding start time $S_{i,od}$ must occur after the casting completion $C_{i,oc}$ plus the minimum curing time θ_i. Resource conflicts are resolved in Equations (4-5): $x_{i,l}$ is a binary variable indicating if component i is assigned to production line l, and $u_{i,j,l}^{line}$ manages the processing sequence between components i and j to prevent temporal overlap. M_{big} is a sufficiently large positive constant.

$$a_i = C_{i,od} + \tau_i, \forall\, i \in I \quad (6)$$

$$l_i \geq a_i - L_i, \forall\, i \in I \quad (7)$$

$$e_i \geq E_i - a_i, \forall\, i \in I \quad (8)$$

$$\sum_t z_{i,t} = 1, \forall\, i \in I \quad (9)$$

$$\sum_t (t\Delta) * z_{i,t} < a_i \leq \sum_t \big((t+1)\Delta\big) * z_{i,t}, \forall\, i \in I \quad (10)$$

$$I_t = I_{t-1} + \sum_i z_{i,t} - D_t, \forall\, t \in T \quad (11)$$

$$0 \leq I_t \leq W_{site}, \forall\, t \in T \quad (12)$$

Equations (6-12) bridge production and on-site assembly. Equation (6) calculates the site arrival time a_i based on the fixed transportation duration τ_i. Equations (7-8) quantify the deviation from the ideal arrival time window $[E_i, L_i]$. To map continuous arrival times to discrete time slots, Equations (9-10) introduce the binary variable $z_{i,t}$, which equals 1 if component i arrives within the time interval t of length Δ. The dynamic equilibrium of on-site inventory is governed by Equations (11-12), where I_t denotes the inventory level at time t, D_t represents the number of consumables used for assembly, and W_{site} is the maximum capacity of the on-site storage yard.

$$shift_i^{time} \geq S_{i,oc} - S_{i,oc}^0, \forall\, i \in I \quad (13)$$

$$shift_i^{time} \geq S_{i,oc}^0 - S_{i,oc}, \forall\, i \in I \quad (14)$$

Finally, Equations (13-14) define the absolute temporal deviation $shift_i^{time}$ by comparing the rescheduled casting start time $S_{i,oc}$ against the original baseline schedule $S_{i,oc}^0$. In summary, the collaborative scheduling model maintains physical and process

feasibility while minimizing deviations from the baseline plan. This provides a solvable MILP formulation for subsequent local rescheduling under event-driven mechanisms.

3 Methods

3.1 Overall Architecture of the Framework

Figure 1 shows the overall architecture of the proposed framework. Within this framework, LLMs are enabling advanced semantic comprehension and causal reasoning upon agent invocation. These agents map dynamic events to specific model patches and executable scripts, synthesizing structured scheduling instructions. To ensure domain alignment, RAG performs semantic similarity searches across specialized databases to retrieve contextually relevant knowledge fragments. This process guides LLMs in generating outputs that strictly adhere to established industry norms and technical constraints[16]. The MAS comprises sense agent (A_{sense}), patch agent (A_{patch}), code agent (A_{code}), scheduling agent ($A_{schedule}$), and dispatch-feedback agent ($A_{dispatch}$). These agents collaborate to form a standardized pipeline, spanning disturbance perception, model updating, code generation, optimization, and instruction dispatching.

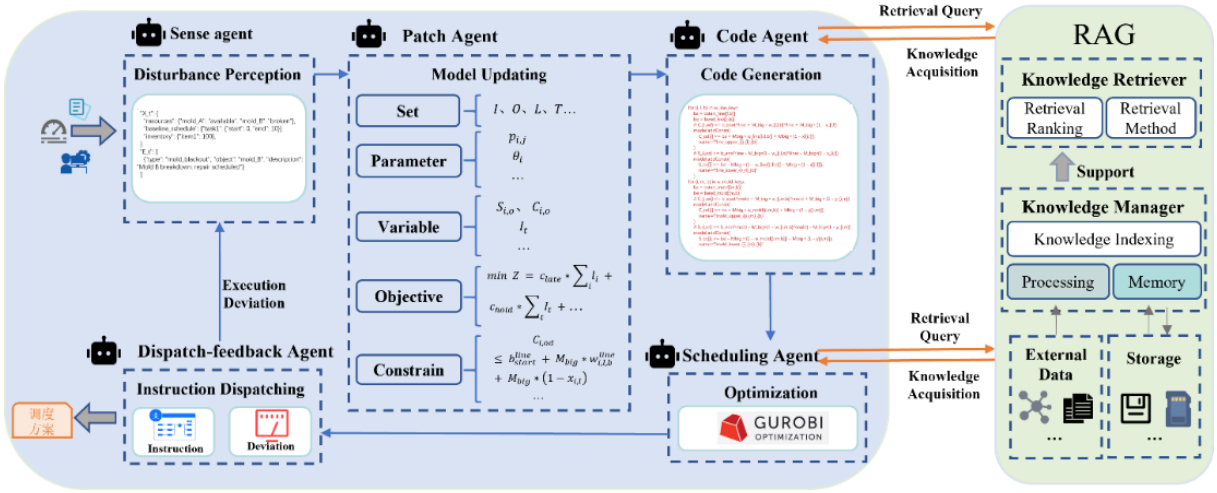

Figure 1. Framework Architecture

Information is transmitted between agents in standardized message formats, collectively completing the management from disturbance perception to instruction dispatching. For any message sent from agent i to agent j, the format is uniformly defined as follows:

$$m_t^{i\to j} = \langle type, payload, t\rangle \tag{15}$$

Here, $type$ specifies the message classification, while $payload$ contains the substantive data encoded as a JSON object. The variable t represents the timestamp, evolving over discrete time horizons $t = 0, 1, 2, \ldots$. Adopting this standardized protocol facilitates modular decoupling and ensures efficient inter-module communication.

3.2 Roles and Functions of AI Agents

The framework comprises five agents, each assigned specific functions and input/output parameters. The rescheduling process is initiated by A_{sense}, which aggregates multi-source data (e.g., operator logs, documents) into production states X_t and disturbance events E_t via function f_{sense}.These structured inputs are transmitted to A_{patch}, as shown in Equations (16-17). A_{patch} maps the disturbances into a mathematical model patch Π_t (Equation 18), which defines parameter variations and constraints, thereby updating the MILP model to M_t (Equation 19).

$$(X_t, E_t) = f_{sense}(NL_t^{req}, docs_t) \tag{16}$$

$$m_t^{sense\to patch} = \langle event\_report, \{X_t, E_t\}, t\rangle \tag{17}$$

$$\Pi_t = \langle \Delta\text{Param}, \text{Blackouts}, \text{NewTasks}, \text{Cancels}, \text{WindowShifts}, \text{Freezes}\rangle = f_{patch}(X_t, E_t) \tag{18}$$

$$M_t = P(M_{t-1}, \Pi_t) \tag{19}$$

$$m_t^{patch\to code} = \langle model\_update, \{M_t\}, t\rangle \tag{20}$$

Following the model update, A_{code} translates the intermediate model M_t into solver-specific executable code $code_t$, accompanied by a static analysis report R_{mod} to ensure syntax validity (Equation 21). This code

drives $A_{schedule}$, which invokes the optimization solver to generate the optimal plan sol_t and diagnostic metrics $Diag_t$ (Equation 23). Finally, $A_{dispatch}$ operationalizes the plan into production instructions and monitors execution feedback $feedback_{t+1}$ (Equation 25), which serves as the input for the subsequent rescheduling iteration (Equation 26).

$$(code_t, R_{mod}) = f_{code}(M_t) \quad (21)$$

$$m_t^{code \to schedule} = \langle MILP\_code, \{code_t, R_{mod}\}, t \rangle \quad (22)$$

$$(sol_t, Diag_t) = f_{schedule}(code_t) \quad (23)$$

$$m_t^{schedule \to dispatch} = \langle plan, \{sol_t, Diag_t\}, t \rangle \quad (24)$$

$$(orders_t, feedback_{t+1}) = f_{dispatch}(sol_t) \quad (25)$$

$$m_{t+1}^{dispatch \to sense} = \langle \text{exec\_feedback}, \{feedback_{t+1}\}, t+1 \rangle \quad (26)$$

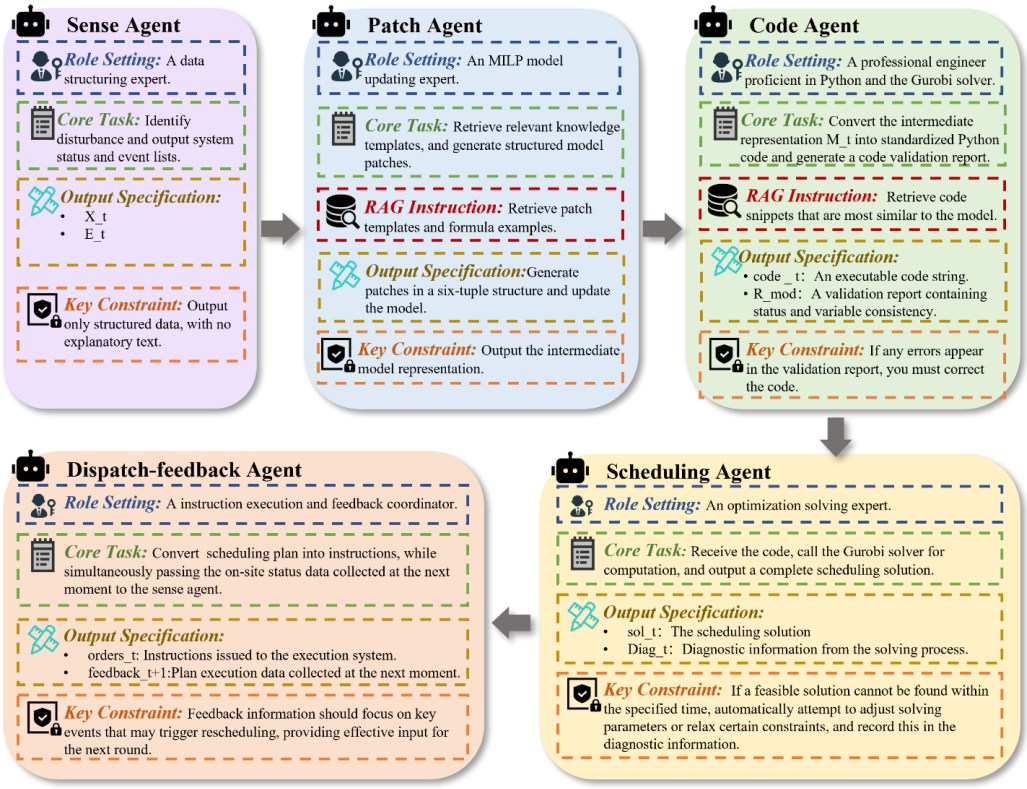

Figure 2. Prompt Structure for Each AI Agent

Prompt engineering enables LLMs to adapt to specific domains through techniques such as task decomposition and example-based guidance[17]. This study designed structured prompts that explicitly specify each agent's role, primary task, output format, and key constraints (see Figure 2). Notably, the prompts for A_{patch} and A_{code} incorporate specific retrieval instructions to leverage external domain knowledge.

3.3 RAG-Based Domain Knowledge Enhancement Mechanism

To ensure factual accuracy and executability, the framework incorporates RAG for A_{patch} and A_{code}. By retrieving relevant domain knowledge via vectorized indexing and injecting it into prompts, RAG mitigates LLM hallucinations and enforces contextual consistency.

3.3.1 Construction of External Knowledge Bases for A_{patch}

The A_{patch} knowledge base maps disturbances to corresponding MILP updates via scenario-patch-MILP pairs. Each entry defines transformation rules for constraints, parameters, and objective functions.

(1) Production Line Unavailability

For disturbances involving production line unavailability, the knowledge base instructs A_{patch} to generate $Blackouts$ patches. A binary auxiliary variable $w_{i,l,b}^{line}$ is introduced to depict the non-overlap logic

between the production line unavailable time $B_l^{line} = [b_{start}^{line}, b_{end}^{line}]$ and production tasks. Constraints (Equations 27-28), active only when $x_{i,l} = 1$, ensure that any tasks assigned to line l do not overlap with the blackout period.

$$C_{i,od} \leq b_{start}^{line} + M_{big} * w_{i,l,b}^{line} + M_{big} * (1 - x_{i,l}), \quad \forall i \in I, \forall l \in L, \forall b \in B_l^{line} \tag{27}$$

$$S_{i,oc} \geq b_{end}^{line} - M_{big} * (1 - w_{i,l,b}^{line}) - M_{big} * (1 - x_{i,l}), \quad \forall i \in I, \forall l \in L, \forall b \in B_l^{line} \tag{28}$$

(2) Emergency Insertion

When an emergency order arrives, it is translated into a $NewTasks$ patch via three update rules: (1) defining the new task set U to expand the total collection to $I^+ = I \cup U$ with initialized decision variables; (2) enforcing the required process and resource constraints for the inserted tasks; and (3) assigning a significantly higher delay penalty w_{i^*} to prioritize urgent insertions. The updated objective function is shown in Equation (29), and the inventory balance constraint is simultaneously extended to I^+, with the updated expression shown in Equation (30).

$$min\ Z = w_{i^*} * \sum_{i^* \in U} l_{i^*} + c_{late} * \sum_{i \in I} l_i + c_{early} * \sum_t e_i + k_{time} * \sum_{i \in I^+} shift_i^{time} + k_{line} * \sum_{i \in I^+} \varphi_i^{line} \tag{29}$$

$$I_t = I_{t-1} + \sum_{i \in I^+} z_{i,t} - D_t, \forall\, t \in T \tag{30}$$

While the preceding discussion primarily addresses disturbances at the production stage, the framework maintains the flexibility to map on-site construction disruptions to specific parameter updates in Equations (1-14). For instance, when encountering exogenous factors such as extreme weather or labor fluctuations, the knowledge base directs A_{patch} to generate a parameter patch ΔParam, reducing the assembly consumption rate D_t for the affected duration and triggers WindowShifts patches to recalibrate the ideal arrival time windows for various components. By continuously indexing disturbances into model parameter updates, the framework ensures its scalability in on-site construction.

3.3.2 Construction of External Knowledge Bases for A_{code}

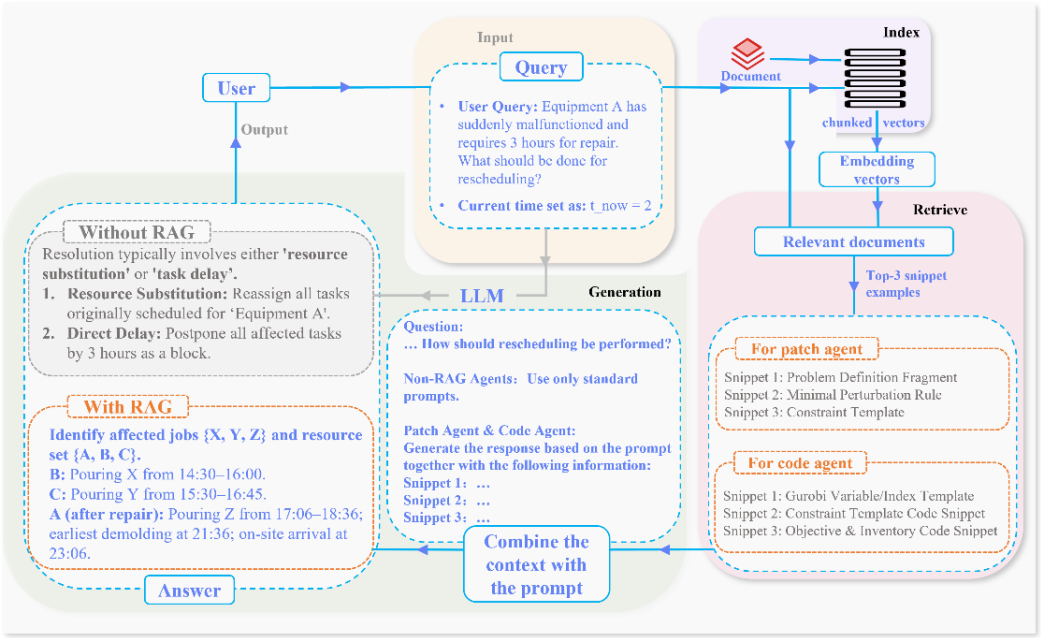

Figure 3. A representative example of the RAG process

The external knowledge base of A_{code} supports the conversion of MILP formulations into executable code by covering common MILP model structures, templates for variable and index definitions, and exemplars for constraint construction and solver interfaces. Given Gurobi's speed and stability for large-scale MILP[18], this study selects Gurobi as the solver for MILP models. To enhance the LLMs' ability to convert formulas into code, the process involves: (1) obtaining normalized MILP expressions from the A_{patch} external knowledge base; (2) driving the LLMs to generate model code using standard prompts without RAG technology; (3) validating executability via the Gurobi solver. This approach avoids semantically correct but non-executable issues that may

arise from relying solely on textual similarity[19-20]. Samples passing this verification are incorporated into the A_{code} knowledge base.

During the reasoning phase, RAG is introduced to reduce the hallucinations in LLM-generated responses and strengthen adherence to domain constraints. When A_{patch} or A_{code} is triggered, the framework retrieves the most contextually relevant template from its corresponding knowledge base. This retrieved knowledge is not memorized by the collaborative scheduling model but dynamically injected as critical contextual information into the original prompt, enhancing generation quality without fine-tuning[17]. The RAG application workflow is illustrated in Figure 3.

Despite the generative capabilities of LLM, the resulting code may encounter execution failures during the solving phase, such as syntax errors, undefined variables, or solver-induced infeasibility. To enhance execution robustness, this framework integrates automated fault diagnosis and repair mechanisms in A_{code} and $A_{schedule}$. In instances of code execution failure, the error message is fed back to A_{code} for context-aware rewriting. Furthermore, when the solver fails to identify a feasible solution, $A_{schedule}$ identifies conflicting constraints and passes the semantic feedback to A_{patch}, which facilitates the dynamic adjustment of non-hard constraints to restore feasibility.

In scenarios where a disturbance is not represented in the existing knowledge base, the framework extracts the original event E_t, the intermediate model M_t, and the diagnostic log $Diag_t$ to generate an anomaly report for expert review. Following manual intervention and verification, the unrecorded disturbance is vectorized and added to the knowledge base. In this way, the knowledge base is continuously expanded through both operational failure records and expert-in-the-loop corrections.

4 Case Study

4.1 Experimental Setup

The experimental environment is implemented in Python 3.9, integrating LangChain for agent orchestration. We employ GPT-4 as the inference model for the AI agents and Gurobi 10.0 as the MILP solver. The case scenario is set in a prefabrication factory containing two parallel production lines (l_1, l_2). The production plan encompasses tasks for facade i_1, composite slab i_2, balcony i_3, air conditioning panel i_4, and staircase i_5. Among these, facades, composite slabs, and staircases are critical path elements, with their late arrival penalty set at a higher rate of 10 units/hour[21]. The early arrival penalty coefficient is set at 50% of the late penalty to reflect that while early arrivals occupy inventory, their consequences are less severe than work stoppages due to material shortages.

The production includes casting, curing, and demolding, with casting and demolding occupying production-line capacity. After completing the casting process, components are transferred via an automated conveyor to the curing chamber. During this period, components are subject only to the minimum curing time constraint and do not occupy production line resources. That is, within the interval $(C_{i,oc}, S_{i,od})$, production line resources can be released for casting or demolding other components. Transportation time is set as $\tau_i = 1.5$ h. Time deviation penalties k_{time} and production line change penalties k_{line} are set at 5 units per hour and 10 units per occurrence, respectively. Other key process parameters are shown in Table 1. For case simplification, only one piece is produced for each component.

Table 1. Component Production Parameters and Logistics Constraints Setting

Component	$p_{i,oc}$ (h)	θ_i (h)	$p_{i,od}$ (h)	c_{late} (units/h)	c_{early} (units/h)
i_1	0.5	10	1.0	10	5
i_2	0.4	10	1.0	10	5
i_3	0.4	10	0.5	8	4
i_4	0.3	10	0.3	8	4
i_5	1.0	10	1.5	10	5

Before production begins, the framework invokes $A_{schedule}$ to construct the MILP model using the aforementioned parameters. All component castings must be intensively completed within the first few hours after t=0 to reserve sufficient curing time. Detailed data for the initial production schedule is shown in Table 2.

Table 2. Initial production schedule

Component	Line	$S_{i,oc} - C_{i,oc}$	$S_{i,od} - C_{i,od}$	a_i	$[E_i, L_i]$
i_1	l_1	0.0-0.5	10.5-11.5	13.0	[12, 15]
i_5	l_1	0.5-1.5	11.5-13.0	14.5	[12, 15]
i_2	l_2	0.0-0.4	10.4-11.4	12.9	[12, 15]
i_3	l_2	0.4-0.8	11.4-11.9	13.4	[12, 15]
i_4	l_2	0.8-1.1	11.9-12.2	13.7	[12, 15]

4.2 Disturbance Scenario Simulation

Production begins at t=0. At t=0.5, the framework receives the natural language log: “Hydraulic system failure on Production Line l_1. Full line shutdown required for 4 hours. Urgent action needed; prioritize staircase delivery.”.

The framework then initiates a collaborative response according to the workflow described in Section 3. A_{sense} parses the log, identifies the affected resource (l_1), the

outage duration, and the impacted task (i_5 casting), and outputs a structured report. A_{patch} generates a mathematical patch Π_t using retrieved knowledge about production line unavailability. The patch freezes completed tasks i_1, imposes fault period constraints on l_1, and reactivates stability penalty terms $k_{time} * \sum_i shift_i^{time} + k_{line} * \sum_i \varphi_i^{line}$ in the objective function. A_{code} converts the patched model into executable code, which is subsequently solved by $A_{schedule}$. Given the high lateness penalty for i_5, $A_{schedule}$ reallocates its casting operation by switching production lines and scheduling it after i_3 and before i_4. Finally, $A_{dispatch}$ is triggered to issue maintenance instructions to l_1 and revised production instructions to l_2.

4.3 Discussion

Table 3 compares a baseline response strategy (Strategy A) with the proposed cross-line rescheduling approach (Strategy B). The results demonstrate that, upon a production-line failure, the proposed framework can parse the disturbance semantics, patch the MILP formulation, and generate an updated production plan.

Strategy B represents an optimized trade-off. By accepting a fixed changeover penalty and small delays for low-priority component i_4, the framework prevents severe delays for critical component i_5. This decision stems from mathematical optimization of the updated model, highlighting the framework's advantage over simple rule-based responses. The end-to-end process completes automatically in under one minute, validating the framework's ability to bridge the semantic-to-model gap and support fully automated rescheduling in practical settings.

Table 3. Full Cost Comparison of Rescheduling Strategies

Cost Item	Strategy A (Wait)	Strategy B (Switch)
$k_{line} * \sum_i \varphi_i^{line}$	0	10.0
$k_{time} * \sum_i shift_i^{time}$	20	6.5
Late arrival penalty of i_5	35.0	0.0
Late arrival penalty of i_4	0.0	1.6
Z	55.0	18.1

5 Conclusion and Future Work

This study proposes an intelligent rescheduling framework that integrates the semantic reasoning of AI agents with the optimization capabilities of MILP. By positioning LLMs as the core cognitive processors, the framework delivers two primary contributions. First, it introduces an automated semantic-to-constraint translation mechanism that autonomously converts unstructured disturbance reports into formal mathematical constraints, thereby accelerating the transition from perception to solution. Second, the framework facilitates localized model refinement, which preserves computational efficiency by bypassing the need for a full model reconstruction.

However, current validation is limited to small-scale disturbance scenarios involving five components. This initial stage serves primarily as a proof-of-concept to demonstrate the feasibility of employing LLM-centric cognitive cores in scheduling workflows. As project scales expand, exact solvers often encounter exponential computational complexity. Therefore, future work should evaluate the framework against large-scale datasets featuring multi-node concurrent disturbances.

Acknowledgement

This study is supported by the National Natural Science Foundation of China (72571252), and the Science and Technology R&D Program of CSCEC International Operations (CSCI-2023-Z-5). Any opinions and findings do not necessarily reflect the views of the grantor.

References

[1]H. Zhong, K. Chen. Strategic planning and control in industrialized construction. *Automation in Construction*, 177: 15, 2025.

[2]J. Zhang, X. Song, E. Diaz. Critical chain project buffer sizing based on resource constraints. *International Journal of Production Research*, 55(3): 671-683, 2017.

[3]M. Hussein, A. E. E. Eltoukhy, A. Karam, I. A. Shaban, T. Zayed. Modelling in off-site construction supply chain management : a review and future directions for sustainable modular integrated construction. *Journal of Cleaner Production*, 310: 32, 2021.

[4]H. Liu, S. Wang, T. Yang, Z. Chen. Optimized transportation scheduling for precast concrete components considering heterogeneous vehicle-size matching. *Advanced Engineering Informatics*, 62: 15, 2024.

[5]A. Mohammadi, M. Tavakolan. Modeling the effects of production pressure on safety performance in construction projects using system dynamics. *Journal of Safety Research*, 71: 273-284, 2019.

[6]Y. Song, J. Wang, J. Lu, X. Si. Research on

collaborative scheduling of multiple projects of prefabricated building based on the niche genetic-raccoon family optimization algorithm. *Alexandria Engineering Journal*, 64: 1015-1033, 2023.

[7]T. van der Beek, J. T. van Essen, J. Pruyn, K. Aardal. Exact solution methods for the resource constrained project scheduling problem with a flexible project structure. *European Journal of Operational Research*, 322(1): 56-69, 2025.

[8]Z. Wang, H. Xu. Scheduling redundancy optimization for precast production to enhance disruption management in prefabricated construction. *Computers & Industrial Engineering*, 192: 15, 2024.

[9]K. Sanogo, A. M. Benhafssa, M. Sahnoun, B. Bettayeb, M. Abderrahim, A. Bekrar. A multi-agent system simulation based approach for collision avoidance in integrated job-shop scheduling problem with transportation tasks. *Journal of Manufacturing Systems*, 68: 209-226, 2023.

[10]G. M. Ali, A. Bouferguene, M. Al-Hussein. Crane mat layout optimization based on agent - based greedy and reinforcement-learning approach. *Journal of Construction Engineering and Management*, 149(8): 17, 2023.

[11]X. Zhang, C. Zhang, J. Sun, J. Xiao, Y. Yang, Y. Luo. EduPlanner : LLM-based multiagent systems for customized and intelligent instructional design. *IEEE Transactions On Learning Technologies*, 18: 416-427, 2025.

[12]Y. Ma, S. Zheng, Z. Yang, P. Zheng, J. Leng, J. Hong. Leveraging large language models in next generation intelligent manufacturing : retrospect and prospect. *Journal of Manufacturing Systems*, 82: 809-840, 2025.

[13]L. Zhao, W. Cheng, L. Meng, C. Zhang, Y. Ren, B. Zhang, P. Duan. MILP modeling and optimization of flexible job shop scheduling problem with preventive maintenance. *Computers & Industrial Engineering*, 201: 16, 2025.

[14]Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. D. Yuxi Bic, J. Sun, Meng, Wang, H. Wang. Retrieval-augmented generation for large language models: a survey. arXiv: 2312.10997, 2024.

[15]K. Shen, X. Li, X. Cao, Z. Zhang. Prefabricated construction process optimization based on rework risk. *Journal of Construction Engineering and Management*, 148(6): 14, 2022.

[16]P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. arXiv: 2005.11401, 2020.

[17]Y. Xie, J. Liu, R. Wang, Z. Wang, K. Yu, Z. Song. Rapid generation method of process routes based on multi-agent collaboration with LLMs. *Advanced Engineering Informatics*, 68: 18, 2025.

[18]B. Zhang, P. Luo, G. Yang, B. Soong, C. Yuen. OR-LLM-agent: automating modeling and solving of operations research optimization problems with reasoning LLM. arXiv:2503.10009, 2025.

[19]J. W. L. Shi, W. Solihin, J. K. W. Yeoh. Fine-tuning a large language model for automated code compliance of building regulations. *Advanced Engineering Informatics*, 68: 103676, 2025.

[20]M. Peng, Z. Chen, J. Yang, J. Huang, Z. Shi, Q. Liu, X. Li, L. Gao. Automatic MILP model construction for multi-robot task allocation and scheduling based on large language models. arXiv:2503.13813, 2025.

[21]J. Yin, R. Huang, H. Sun, S. Cai. Multi-objective optimization for coordinated production and transportation in prefabricated construction with on-site lifting requirements. *Computers & Industrial Engineering*, 189: 15, 2024.