https://pubsonline.informs.org/journal/ited

INFORMS TRANSACTIONS ON EDUCATION

Vol. 00, No. 0, Xxxxx 0000, pp. 000–000

Submitted to *INFORMS Transactions on Education*

ISSN 1532-0545, EISSN 1532-0545

Generative AI as Scaffolding for Operations Research Education: Framework, Classroom Evaluation, and Design Principles (Supplementary Material)

Luis Fernando PÉREZ ARMAS

IESEG School of Management, Univ. Lille, CNRS, UMR 9221 - LEM- Lille Economie Management, F-59000 Lille, France. l.perezarmas@ieseg.fr

Stephania KOSSMAN

SAVILA EDUCATION, 679 Avenue de La Republique, Lille, F-59800, Nord, France. hello@savilaeducation.com

Authors are encouraged to submit new papers to INFORMS journals by means of a style file template, which includes the journal title. However, use of a template does not certify that the paper has been accepted for publication in the named journal. INFORMS journal templates are for the exclusive purpose of submitting to an INFORMS journal and are not intended to be a true representation of the article's final published form. Use of this template to distribute papers in print or online or to submit papers to another non-INFORM publication is prohibited.

Abstract.

Operations Research education requires students to progress through a demanding sequence of cognitive transitions, from problem framing and mathematical formulation to computational implementation, validation, and interpretation, yet scalable instructional support for these transitions remains limited. This article proposes a task-centered framework that maps generative AI (GenAI) support functions to the main bottlenecks of OR learning and specifies the instructor role needed at each stage. We illustrate the framework through six use cases, organized by implementation complexity and evaluated in terms of pedagogical role and risks. We then evaluate one classroom implementation in depth: a study in which 66 master's students used a customized GPT companion to implement and solve a quadratic multi-knapsack case in Python and Pyomo. Mixed-methods evidence from pre/post surveys, regression and mediation analyses, a compact path model, and qualitative coding shows that the intervention worked through manageability rather than task ease. Confidence gains were large and concentrated in the computational stages of OR modeling ($d_z = 1.29$ overall). The findings suggest that GenAI is most valuable in OR education not when it removes difficulty, but when it makes difficult modeling and implementation tasks manageable enough for students to engage, persist, and learn.

Key words: Operations Research Education, Generative AI, Instructional Scaffolding, Mathematical Programming, Task Manageability

Appendix A: Supplementary Examples for GenAI Use Cases

This appendix provides concrete examples for the GenAI use cases discussed in Section 3. All prompts, scripts, and full implementation materials are available in the GitHub Repository (`https://github.com/ceche1212/GenAI_for_OR_education`).

A.1. Audio-Based Learning Example

Table 1 presents selected exchanges from a NotebookLM-generated podcast on the Order Batching Problem (Pardo et al. 2024). The excerpt illustrates how a conversational audio format can introduce students to the operational context and economic stakes of an OR problem before formal notation is introduced. The supplementary material includes the prompts used to generate this content, a Python script for extracting transcripts from generated audio, and two complete podcast examples covering the Order Batching Problem and Simulated Annealing. Instructors who prefer greater control over voice and pacing may generate a script using any LLM and convert it to audio using an external text-to-speech service.

Speaker	**Text**
Speaker A	*"You've just clicked buy now online, that satisfying moment when your order is confirmed."*
Speaker B	*"But have you ever really thought about what happens between that click and the package arriving at your door?"*
Speaker A	*"You know, beyond the shipping label?"*
Speaker B	*"Yeah, there's this whole hidden world, isn't there? A world of warehouses, picking lists, and optimization puzzles."*
Speaker A	*"An incredibly complex optimization process, in fact, where grouping customer orders efficiently, what researchers call* order batching*, can make the difference between smooth operations and costly delays."*
⋮	⋮
Speaker A	*"Digging through this research, one number really jumped out at me. It said implementing order batching can cut warehouse travel time by up to 35%. That's a huge number, isn't it?"*
Speaker B	*"It's staggering. Is there a catch, a trade-off, or is it just pure efficiency gain? And how does that affect the price I pay?"*
Speaker A	*"It really is a remarkable gain. Sure, there might be a little more complexity in sorting the items out after picking, depending on how you do it. But the reduction in travel time usually blows any minor downsides out of the water."*
Speaker B	*"And when you think about the money involved, wow. The picking process itself, just finding and grabbing items, is a massive cost for warehouses. This review suggests it can be up to 60% of all labor activities, maybe more than half of total operational costs. So if you can slash travel time by 35% for something that costs that much, it's huge for the bottom line."*

Table 1 Selected exchanges from a NotebookLM-generated podcast on the Order Batching Problem (Pardo et al. 2024), illustrating how conversational audio can introduce students to OR problem contexts before class.

A.2. Flashcards for OR Modeling Patterns

Table 2 shows a sample of flashcards generated using Google AI Studio (Gemini 2.5) from a set of integer programming modeling techniques. Each card pairs a question requiring active recall with a concise, notation-precise answer. The supplementary material includes prompt templates for generating both basic cards and cloze-style cards, in which the learner completes a missing expression within a statement, for example by identifying that $x_i + x_j \leq 1$ models mutual exclusivity. Both formats are compatible with common flashcard platforms such as Anki and Quizlet.

Front (Question)	**Back (Answer)**
How can you use integer programming to constrain a variable x to be **odd**?	Introduce a new integer variable $w \geq 0$ and write: $x - 2w = 1$.
How do you model the congruence relation $x \equiv a \pmod{b}$ in integer programming?	Introduce a new integer variable w and write: $x - bw = a$.
How do you restrict a variable x to take a value from a discrete set, for example $x \in \{4, 8, 13\}$?	Introduce binary variables w_1, w_2, w_3 and write: (1) $x = 4w_1 + 8w_2 + 13w_3$, and (2) $w_1 + w_2 + w_3 = 1$.
What is the big-M method in integer programming?	A technique for modeling logical and conditional constraints using a binary variable and a large constant M chosen to exceed any feasible value of the constraint expression.
How do you model the constraint "exactly one of items i, j, k is selected"?	Write $x_i + x_j + x_k = 1$. Since the variables are binary, this forces exactly one to equal 1.

Table 2 Sample flashcards for integer programming modeling techniques, generated using Google AI Studio (Gemini 2.5). Cards are formatted for direct import into Anki or Quizlet.

A.3. Prompt Templates for AI Tutors, Dialogue Agents, and Case-Study Companions

The GitHub Repository provides prompt templates for three types of customized learning companions: Socratic OR tutors, adversarial dialogue agents, and case-study companions. The Socratic tutor guides students through problem framing, formulation, implementation, and interpretation without providing complete solutions. The adversarial agent challenges assumptions, requests justification, and prompts students to identify weaknesses in their reasoning. The case-study companion combines both roles, guiding students through an instructor-designed trajectory while requiring them to make and defend decisions at each step.

All templates are designed to be adapted across OR topics, modeling environments, solver interfaces, and case formats. Instructors can adjust the level of guidance, the types of hints permitted, the interaction tone, and the degree to which the companion may provide code, formulations, or diagnostic feedback. The central design constraint in all cases is that the companion supports student reasoning rather than replacing it.

Table 3 illustrates a short exchange with the boxer-coach companion created for the case *Resource Planning at L'Audobarquoise of Saint-Omer: A Graph Theory and Mathematical Programming Case Study with Python* (PÉREZ ARMAS 2024). The exchange shows the companion alternating between coaching and challenge, asking the student to frame the optimization problem and justify a modeling choice.

Speaker	**Text**
Companion	*"Welcome aboard, strategist! Before we start sparring, let us ground ourselves: what is the optimization problem in one sentence?"*
Student	*"Mr. Lauweriere wants to minimize the number of boats he needs to cover the booked trips."*
Companion	*"Excellent start. You have identified the main bottleneck: the boats. But let us push a little: what about overlapping tours and limited employees? Should staffing be part of the objective, or treated as a constraint?"*
Student	*"My instinct tells me it should be a constraint, that we do not assign a resource to overlapping trips."*

Table 3 Sample interaction with the boxer-coach case-study companion for the L'Audobarquoise case (PÉREZ ARMAS 2024), illustrating how the companion alternates between coaching and challenge.

A.4. Examples for Higher-Complexity GenAI Supports

The following examples illustrate possible implementations of the higher-complexity use cases discussed in Section 3.4. They are intended as starting points for adoption rather than prescriptions for a single technical architecture. Additional background on the underlying architectures is provided in Appendix B.

A.4.1. Retrieval-Augmented Knowledge Assistants A retrieval-augmented knowledge assistant grounds GenAI responses in a curated document collection, reducing hallucination and improving the specificity of guidance. For OR education, relevant knowledge bases might include mathematical programming textbooks for formulation support (Sánchez 2022, Williams 2013), specialized domain references such as project scheduling handbooks and papers on the Resource-Constrained Project Scheduling Problem (Artigues et al. 2008, Demeulemeester and Herroelen 2002, Schiwindt and Zimmermann 2015, Schwindt and Zimmermann 2015, Vanhoucke et al. 2012), and solver manuals, modeling-language documentation, and validated course notebooks for implementation support.

The supplementary material includes a simple retrieval-augmented generation prototype built with LangChain that instructors can adapt to connect an LLM to course-specific documents. It is intended as an illustrative starting point rather than a production-ready system. As a publicly available external example, Gurobi provides Gurobot, a retrieval-augmented assistant for querying Gurobi and gurobipy documentation. Analogous systems could be built for other solvers, modeling languages, or course repositories.

A.4.2. Structured Output Example for Adapted Models Adapted models can generate OR teaching materials in a consistent, structured format, reducing instructor preparation time and supporting scalable assessment design. The JSON example below illustrates how a problem instance for the Capacitated Vehicle Routing Problem (CVRP) can be represented through sets, parameters, variables, objectives, constraints, and notes. This structure can be used as a prompting template, a fine-tuning target, or a validation schema for generated outputs.

```
{
  "problem_family": "CVRP",
  "sets": [
    {"name":"V","desc":"set of all nodes {0,...,n} with depot 0"},
    {"name":"C","desc":"customer nodes V \\ {0}"}
  ],
  "parameters": [
    {"name":"d_i","desc":"demand of customer i",
     "domain":"Z_+","indices":"i in C"},
    {"name":"Q","desc":"vehicle capacity","domain":"Z_+"}
  ],
  "variables": [
    {"name":"x_{ij}","desc":"1 if arc (i,j) used",
     "domain":"{0,1}","indices":"i,j in V, i!=j"},
    {"name":"u_i","desc":"MTZ load variable",
     "domain":"R_+","indices":"i in C"}
  ],
  "objective": {"sense":"min","expr":"sum_{i!=j} c_{ij} x_{ij}"},
  "constraints": [
    {"label":"visit","expr":"sum_{j!=i} x_{ij} = 1",
     "indices":"i in C"},
    {"label":"leave","expr":"sum_{j!=i} x_{ji} = 1",
     "indices":"i in C"},
    {"label":"depot_degree","expr":
```

```
      "sum_{j in C} x_{0j} = k = sum_{j in C} x_{j0}"},
    {"label":"capacity_MTZ","expr":
      "u_i - u_j + Q x_{ij} <= Q - d_j",
      "indices":"i in C, j in C, i!=j"},
    {"label":"bounds","expr":"d_i <= u_i <= Q",
      "indices":"i in C"}
  ],
  "notes": [
    "Use MTZ subtour elimination.",
    "If k unknown, allow multiple vehicles implicitly."
  ]
}
```

A.4.3. Illustrative Agentic OR Learning Environment Figure 1 illustrates a speculative but pedagogically motivated agentic OR learning environment, in which specialized agents coordinate problem interpretation, formulation, method selection, implementation, solving, validation, and reporting. The figure is included to clarify the design horizon discussed in Section 3.4 and should not be interpreted as an evaluated classroom system.

Appendix B: Generative AI Architectures: RAG, LoRA, and Agentic Systems

Generative Artificial Intelligence refers to machine learning systems capable of producing novel outputs such as text, images, code, or mathematical models, conditioned on input prompts. These systems are often referred to as Large Language Models (LLMs) when dealing with text, mathematical, or code-based tokens. They are typically implemented using large-scale neural network architectures, most commonly Transformer-based models Vaswani et al. (2017), trained on vast corpora of text or multimodal data. Training is performed via next-token prediction (causal language modeling) or related objectives, where the model learns a probability distribution over sequences of tokens. Through exposure to diverse language and domain-specific patterns, the model internalizes both syntactic structure and semantic associations, enabling it to generate coherent, contextually relevant continuations for new inputs.

From a technical perspective, a generative language model can be represented as a parameterized function $f_\theta : \mathcal{X} \to \mathcal{Y}$ mapping token sequences to probability distributions over the next token. The parameters θ, often numbering in the billions, are optimized via stochastic gradient descent to minimize cross-entropy loss against a training dataset. Architecturally, the Transformer leverages self-attention mechanisms to capture long-range dependencies and contextual relationships, allowing the model to dynamically weight different parts of the input when generating each output token.

Although such models exhibit impressive generalization capabilities, they are limited by the knowledge contained in their static training data, and fine-tuning them for domain-specific applications can be computationally expensive. This has motivated the development of architectural extensions and parameter-efficient adaptation methods. Three paradigms of particular relevance are discussed below, and their workflow is shown in detail in Figure 2.

Retrieval-Augmented Generation (RAG). RAG addresses the knowledge limitation of static LLMs by integrating an external retrieval component into the generation process (Lewis et al. 2020). At inference time, a retriever module queries a vector database or search index to obtain relevant documents based on the user input. In practice, for efficiency, especially when the text corpus is large, the context database is preprocessed and split into smaller, semantically coherent chunks, each of which is vectorized using textual embeddings derived from the same or a compatible LLM. These embeddings encode the semantic content of each chunk into a high-dimensional vector space. When a user query x is received, it is similarly transformed into an embedding, and a nearest-neighbor search is performed over

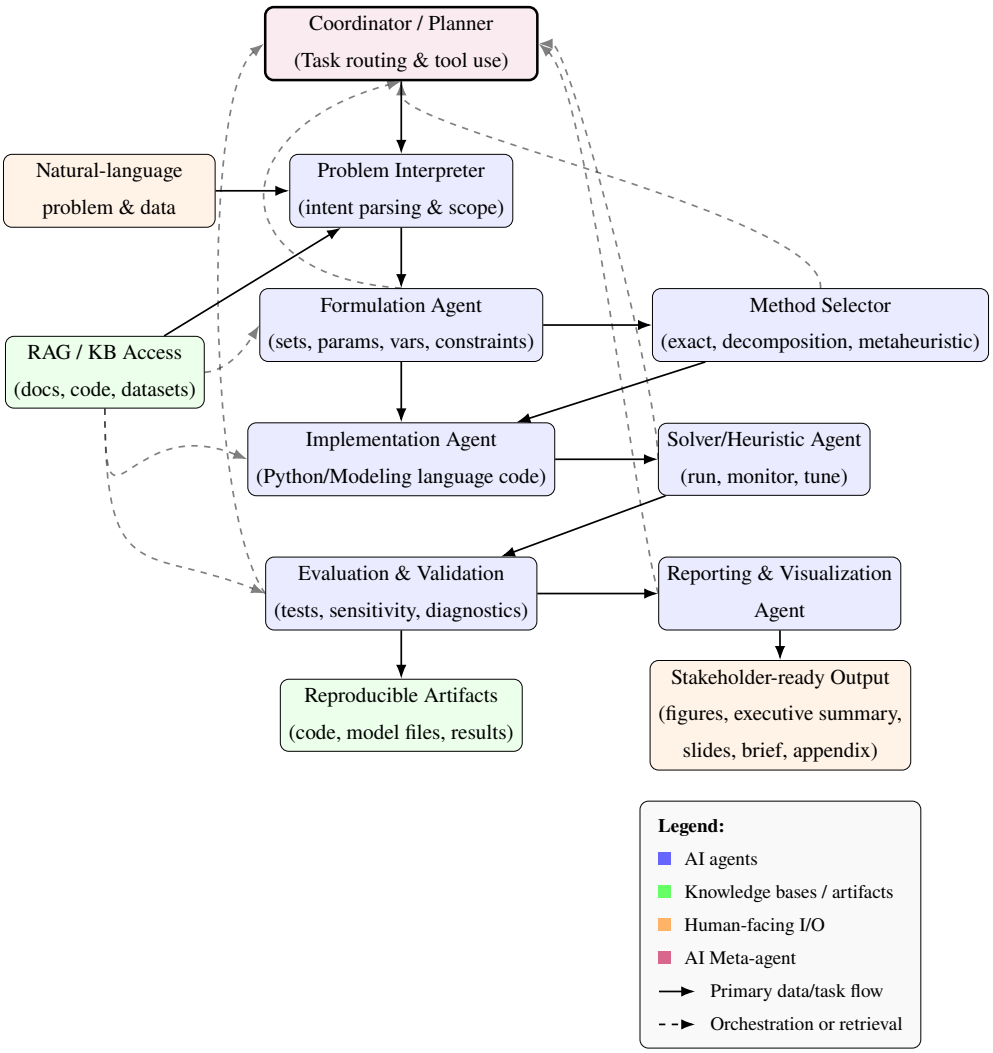


Figure 1 **Illustrative agentic OR modeling assistant. Specialized agents coordinate problem interpretation, formulation, method selection, implementation, solving, validation, and reporting, with retrieval-augmented access to domain repositories. This figure represents a design horizon rather than an evaluated system.**

the vector database. The search typically returns the top-k vectors with the smallest cosine distance (or, equivalently, the highest cosine similarity) to the query embedding, corresponding to the most semantically relevant chunks in the corpus. These retrieved passages are then injected into the model's context window before generation. The process can be summarized as:

$$x \xrightarrow{\text{embed + retrieve}} \{d_1, \ldots, d_k\} \xrightarrow{\text{LLM}} y$$

where x is the input query, $\{d_i\}$ are retrieved documents, and y is the generated output. By grounding the LLM's generation in this retrieved context, RAG improves factual accuracy, reduces hallucination[1], enables knowledge updates without retraining, and provides verifiable outputs via cited sources. This architecture is presented in Figure 3 below.

Low-Rank Adaptation (LoRA). Low-Rank Adaptation or LoRA (Hu et al. 2022) is a technique for fine-tuning large pre-trained language models without having to update all of their parameters. Instead of modifying the original

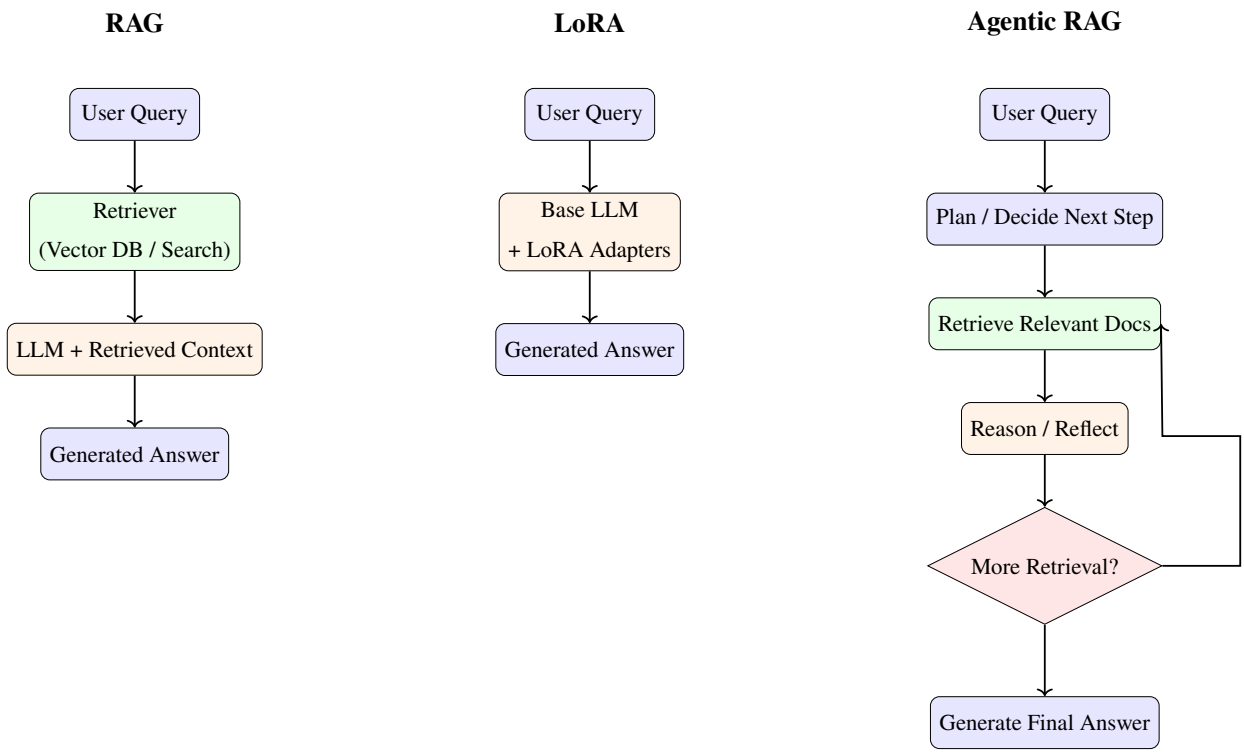


Figure 2 Comparison of RAG, LoRA, and Agentic RAG workflows. RAG augments an LLM with retrieved documents at inference time; LoRA adapts model parameters efficiently via low-rank matrices; Agentic RAG uses iterative planning, retrieval, and reasoning loops for complex tasks.

(and very large) weight matrices of the Transformer layers, LoRA keeps those weights frozen and adds a pair of much smaller, trainable matrices that act as an "adjustment layer."

Formally, suppose a Transformer layer contains a weight matrix $W_0 \in \mathbb{R}^{d\times k}$. LoRA re-parameterizes it as:

$$W = W_0 + BA$$

where $B \in \mathbb{R}^{d\times r}$ and $A \in \mathbb{R}^{r\times k}$, with $r \ll \min(d, k)$ known as the *rank* of the adaptation. Because r is small, the number of trainable parameters in A and B is tiny compared to the full size of W_0. This workflow is presented in Figure 4 below.

In practical terms, this means the model can be specialized to a new domain or task by learning only these small low-rank updates, rather than retraining the entire model. The benefits are substantial: fine-tuning becomes faster, requires less memory, and can often be done on a single GPU, all while preserving the original model's general language understanding capabilities.

Agentic RAG and Integrated Architectures. Standard RAG pipelines perform a single retrieve-then-generate step. Agentic RAG extends this by embedding the retrieval process within an autonomous reasoning loop. Here, the model acts as an "agent" that can plan multi-step actions, call external tools, reformulate queries, and iteratively refine retrieved context before producing a final answer (Singh et al. 2025). This architecture as shown in Figure 5, often employs control loops such as Plan → Retrieve → Reason/Reflect → Generate → Repeat if needed.

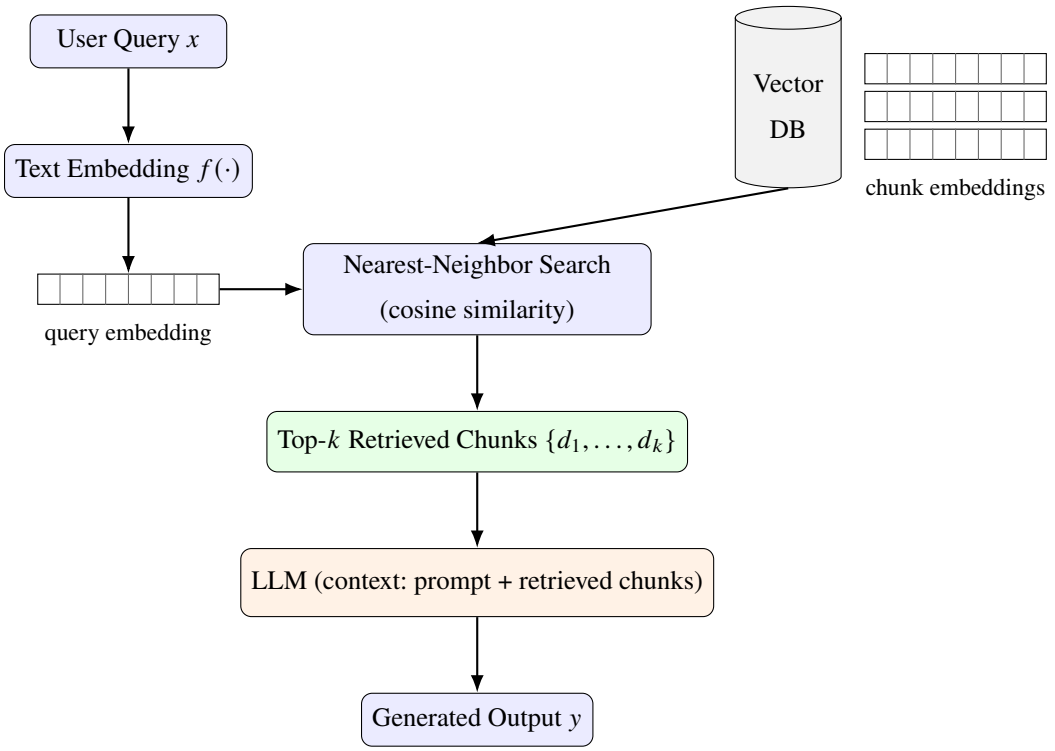


Figure 3 **RAG workflow: the query is embedded and compared (via cosine similarity) against a vector database of chunk embeddings; the top-k nearest chunks are injected into the LLM's context window to ground generation.**

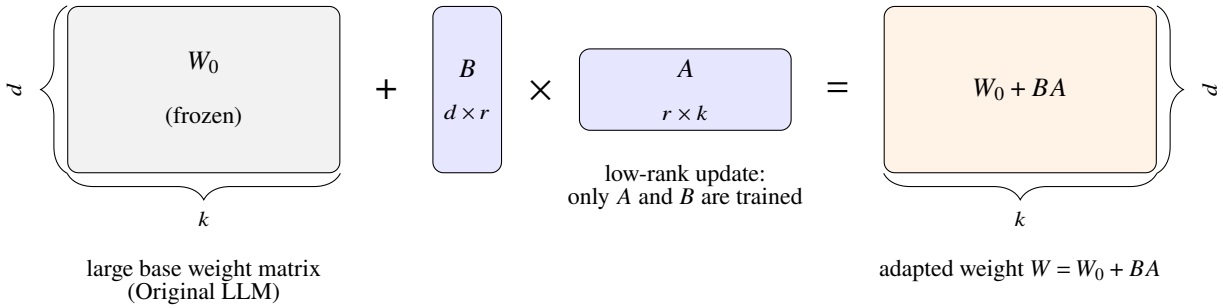


Figure 4 **LoRA schematic. The large base matrix W_0 is *frozen*. A small low-rank update BA (with $r \ll \min(d, k)$) is learned and added to W_0, yielding $W = W_0 + BA$.**

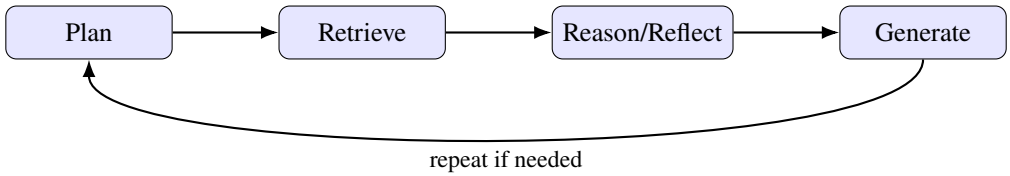


Figure 5 **Description of an Agentic loop architecture.**

This allows agentic systems to handle more complex, multi-faceted queries. A particular distinctive feature of agentic systems is their ability to interact with external tools, such as search engines, databases, APIs, code interpreters, or optimization solvers, and to be trained or configured to select the most appropriate tool for each subtask. Importantly, a "tool" in this context could itself be another agent, enabling the design of hierarchical or distributed multi-agent systems in which specialized agents collaborate, delegate tasks, and share intermediate results. This compositional design opens the possibility of building sophisticated ecosystems of cooperating agents, each with distinct capabilities, to solve large, complex problems that exceed the capacity of any single model.

Further integrations combine RAG with LoRA fine-tuning, knowledge graphs, or re-ranking mechanisms to maximize retrieval precision and contextual relevance. These architectural innovations, namely RAG for dynamic knowledge grounding, LoRA for efficient domain adaptation, and Agentic RAG for autonomous, iterative reasoning with tool use, form a versatile toolkit for extending the capabilities of generative models beyond the constraints of their base training and enable precise, up-to-date, and context-aware generation.

Appendix C: Questionnaires

This appendix reports the full pre- and post-survey instruments used in the classroom evaluation. Items 1–4 of the pre-survey capture factual prior experience and were measured on binary or ordinal scales. Items 5–27 were measured on five-point Likert scales unless otherwise noted. The post-survey follows the same convention. Paired confidence items appear in both surveys and are indicated in brackets. Reverse-coded items are marked with an asterisk.

C.1. Pre-Survey Questionnaire

Prior experience

1. In the previous decision modeling or OR course, what was your final grade range?
2. Before this session, have you ever implemented an optimization model in Python?
3. Before this session, have you ever used Pyomo?
4. Before this session, have you ever used an optimization solver through Python, such as HiGHS, Gurobi, CPLEX, or CBC?

OR modeling proficiency *(paired with post-survey items 17–21)*

5. I can translate a business problem into decision variables.
6. I can formulate an objective function for an optimization problem.
7. I can formulate constraints for an optimization model.
8. I can understand a mathematical optimization model written with formal notation.
9. I feel confident explaining the meaning of decision variables, constraints, and objective functions in an OR model.

Programming proficiency *(paired with post-survey items 18–20)*

10. I am comfortable programming in Python.
11. I can read and understand Python code written by someone else.
12. I can modify existing Python code to solve a new problem.
13. I can debug Python code when it produces errors.
14. I feel confident implementing mathematical models in Python.

Generative AI proficiency and verification tendency

15. Before this session, how often have you used generative AI tools for academic work?
16. Before this session, how often have you used generative AI tools for programming or coding help?
17. I know how to write prompts that usually produce useful answers from generative AI tools.
18. I know how to evaluate whether an answer produced by generative AI is reliable.
19. Generative AI can produce answers that sound correct even when they are wrong.
20* When a generative AI tool gives a plausible answer, I tend to accept it quickly. *(reverse-coded)*
20. I usually verify the answers produced by generative AI before relying on them.

Perceived value of OR

22. Generative AI tools can be useful for learning technical subjects.
23. Generative AI tools can help students implement optimization models in code.
24. Learning operations research and decision modeling is valuable for business decision-making.
25. Skills in optimization and mathematical modeling are useful for my future career.
26. Learning to implement optimization models in Python is professionally valuable.
27. I am motivated to improve my ability to formulate and implement optimization models.

Open-ended

28. In your experience, what is usually the most difficult aspect of OR modeling or optimization model implementation?

C.2. Post-Survey Questionnaire

Performance indicators *(self-reported)*

1. Did you manage to solve the case study? Solving means finding a feasible solution that satisfies all problem constraints.
2. Please enter your best objective function value. If you did not solve the case, enter -1.
3. How many times did you re-run your notebook to obtain a better solution?
4. How much time did it take you to obtain a feasible solution?

Support usage

5. Did you listen to the pre-class podcast before the session?
6. How much did you use the GPT companion during the activity?
7. For which purposes did you use the GPT companion? *(select all that apply)*
8. How much did you rely on the instructor's introductory example during the activity?
9. How actively did you use the case study text during the activity?

Perceived learning gain

10. This activity improved my understanding of the optimization problem.
11. This activity improved my understanding of the mathematical formulation.
12. This activity improved my ability to translate a mathematical optimization model into Pyomo code.
13. This activity improved my ability to debug optimization code.
14. This activity improved my ability to interpret solver output.
15. This activity improved my ability to extract and report the solution correctly.
16. This activity helped me better understand the link between mathematical modeling and implementation.

Confidence *(paired with pre-survey items 9, 10, 13, 14, and 21)*

17. After this activity, I feel confident explaining the meaning of decision variables, constraints, and objective functions in an OR model.
18. After this activity, I am more comfortable programming in Python.
19. After this activity, I am more comfortable debugging Python code when it produces errors.
20. After this activity, I feel confident implementing mathematical models in Python.
21. After this activity, I feel confident using generative AI as a support tool for learning.
22. After this activity, I feel more confident tackling similar optimization implementation tasks.
23. After this activity, I feel more confident interpreting the output of an optimization solver.

Difficulty and effort

24. I found this activity intellectually challenging.
25. I found the implementation phase difficult.
26. I found extracting the solution difficult.
27. I had to invest substantial effort to complete the activity.

Task manageability and integrated support

28. The support available during the activity made the task manageable.
29. The activity helped me work more autonomously than I otherwise could have.

GPT companion evaluation

30. The GPT companion provided clear explanations.
31. The GPT companion was useful for understanding the mathematical model.
32. The GPT companion was useful for translating the model into code.
33. The GPT companion was useful for debugging.
34. The GPT companion sometimes gave misleading or incorrect guidance.
35. I had to verify the GPT companion's answers carefully.
36* I relied too much on the GPT companion during the activity. *(reverse-coded)*
36. The GPT companion reduced my frustration during the activity.
37. The GPT companion helped me progress faster than I otherwise would have.

Instructor support

39. The pre-class podcast helped me approach the problem with better initial understanding.
40. The instructor's introductory example helped me prepare for the main task.
41. The instructor debrief helped consolidate my understanding after the activity.
42. Overall, the combination of instructor guidance and GPT support was effective for this learning activity.

Administrative

43. Please provide the shareable link to your conversation with the GPT companion.

Overall satisfaction

44. Overall, I am satisfied with this learning activity.

Open-ended

45. What aspect of the activity helped your learning the most, and why?
46. What aspect of the activity was most difficult or least helpful, and why?
47. If you used the GPT companion, in what way was it most useful or least useful?

Appendix D: Descriptive Statistics

This appendix reports the sample profile, construct reliability statistics, and the full correlation matrix for the main variables used in the analyses. Table 4 summarizes participant characteristics and activity outcomes. Table 5 reports Cronbach's alpha for all multi-item constructs. Figure 6 displays the correlation matrix as a heatmap, and Table 6 reports the full numerical values.

Indicator	**Value**
Sample	
Matched analytical sample	$N = 66$
Total enrolled students	$N = 73$
Prior OR/decision-modeling grade, mean (SD)	12.62/20 (2.98)
Prior technical experience (ordinal scales; higher = more experience)	
Prior Python optimization implementation, mean (median)	1.15 (1.00)
Prior Pyomo use, mean (median)	0.12 (0.00)
Prior solver use through Python, mean (median)	0.76 (0.50)
Activity outcomes	
Self-reported solved case, *n* (%)	53 (80.3%)
Best observed objective value	232.20
Standardized solution quality, mean (SD)	0.605 (0.366)
Attempts among feasible students, mean (median)	2.08 (2.00)
Time to feasibility among feasible students, mean (SD)	41.2 (16.0) min
Support resource usage (5-point scale; higher = more use)	
Podcast use, mean (median)	1.27 (1.00)
GPT companion use, mean (median)	2.33 (2.00)
Instructor example use, mean (median)	2.09 (2.00)
Case text use, mean (median)	1.67 (2.00)

Table 4 Sample profile and activity outcomes. Solution quality is defined as the student's reported objective value divided by the best observed feasible objective value across the matched sample.

Construct	k	**Cronbach's α**
Pre-survey constructs		
OR proficiency	5	0.869
Programming proficiency	5	0.857
LLM proficiency	6	0.571
OR value	4	0.787
Verification tendency	3	0.438
Post-survey constructs		
Perceived learning gain	7	0.868
Difficulty / effort	4	0.407
GPT usefulness	6	0.747
GPT risk / overreliance	2	0.578
Instructor support	2	0.513

Table 5 Construct reliability for the matched sample ($N = 66$). All constructs were measured on five-point Likert scales. k denotes the number of items. Constructs with $\alpha < 0.6$ are interpreted with additional caution in the main analyses.

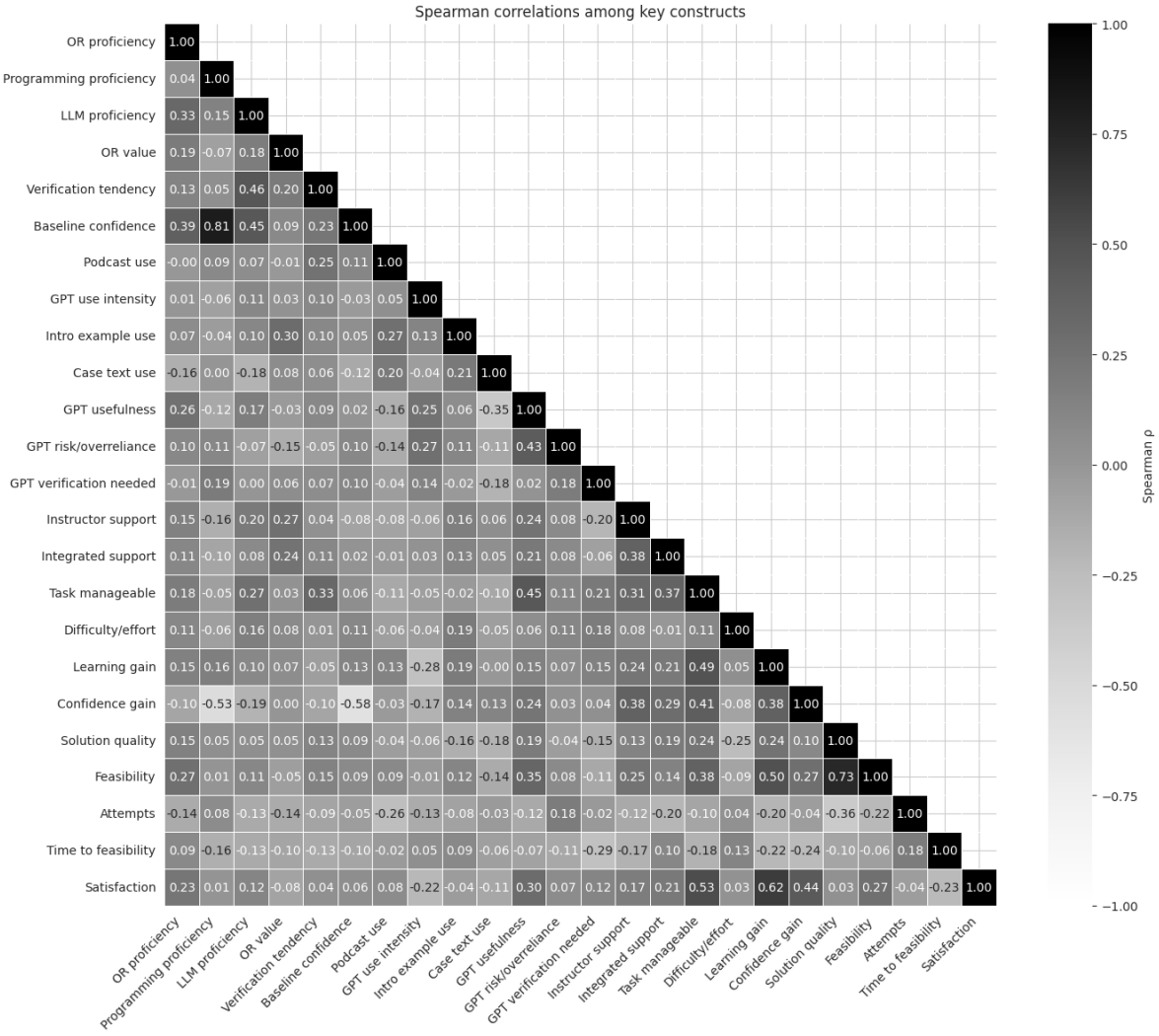


Figure 6 Spearman correlation matrix for main variables, displayed as a heatmap. Darker shading indicates stronger correlations. Numerical values are reported in Table 6.

Variable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1. OR proficiency																							
2. Programming proficiency	.04																						
3. LLM proficiency	.33**	.15																					
4. OR value	.19	-.07	.18																				
5. Verification tendency	.13	.05	.46**	.20																			
6. Baseline confidence	.39**	.81**	.45**	.09	.23																		
7. Podcast use	-.00	.09	.07	-.01	.25*	.11																	
8. GPT use intensity	.01	-.06	.11	.03	.10	-.03	.05																
9. Intro example use	.07	-.04	.10	.30*	.10	.05	.27*	.13															
10. Case text use	-.16	.00	-.18	.08	.06	-.12	.20	-.04	.21														
11. GPT usefulness	.26*	-.12	.17	-.03	.09	.02	-.16	.25*	.06	-.35**													
12. GPT risk/overreliance	.10	.11	-.07	-.15	-.05	.10	-.14	.27*	.11	-.11	.43**												
13. GPT verification needed	-.01	.19	.00	.06	.07	.10	-.04	.14	-.02	-.18	.02	.18											
14. Instructor support	.15	-.16	.20	.27*	.04	-.08	-.08	-.06	.16	.06	.24*	.08	-.20										
15. Integrated support	.11	-.10	.08	.24*	.11	.02	-.01	.03	.13	.05	.21	.08	-.06	.38**									
16. Task manageable	.18	-.05	.27*	.03	.33**	.06	-.11	-.05	-.02	-.10	.45**	.11	.21	.31*	.37**								
17. Difficulty/effort	.11	-.06	.16	.08	.01	.11	-.06	-.04	.19	-.05	.06	.11	.18	.08	-.01	.11							
18. Learning gain	.15	.16	.10	.07	-.05	.13	.13	-.28*	.19	-.00	.15	.07	.15	.24	.21	.49**	.05						
19. Confidence gain	-.10	-.53**	-.19	.00	-.10	-.58**	-.03	-.17	.14	.13	.24	.03	.04	.38**	.29*	.41**	-.08	.38**					
20. Solution quality	.15	.05	.05	.05	.13	.09	-.04	-.06	-.16	-.18	.19	-.04	-.15	.13	.19	.24	-.25*	.24*	.10				
21. Feasibility	.27*	.01	.11	-.05	.15	.09	.09	-.01	.12	-.14	.35**	.08	-.11	.25*	.14	.38**	-.09	.50**	.27*	.73**			
22. Attempts	-.14	.08	-.13	-.14	-.09	-.05	-.26*	-.13	-.08	-.03	-.12	.18	-.02	-.12	-.20	-.10	.04	-.20	-.04	-.36**	-.22		
23. Time to feasibility	.09	-.16	-.13	-.10	-.13	-.10	-.02	.05	.09	-.06	-.07	-.11	-.29*	-.17	.10	-.18	.13	-.22	-.24	-.10	-.06	.18	
24. Satisfaction	.23	.01	.12	-.08	.04	.06	.08	-.22	-.04	-.11	.30*	.07	.12	.17	.21	.53**	.03	.62**	.44**	.03	.27*	-.04	-.23

Table 6 **Spearman correlation matrix for main variables ($N = 66$). The table reports the lower triangle.** $^{*}p < .05$, $^{**}p < .01$.

Appendix E: Detailed Results

This appendix reports the full outputs for the mediation analyses, the compact path model, and the qualitative coding procedure. Results are organized to correspond directly with the analyses reported in Sections 5.3, 5.4, and 5.5.

E.1. Mediation Analysis Outputs

Table 7 reports the full outputs for the five exploratory mediation models estimated in Sections 5.3 and 5.4. All variables were standardized before estimation. Confidence intervals are bootstrap confidence intervals based on 5,000 resamples. Each panel reports the first-stage path (predictor to mediator or predictor to intermediate outcome), the second-stage path (mediator to outcome), and the total, direct, and indirect effects.

Mediation model	Path	β	SE	p	CI 2.5%	CI 97.5%
GPT usefulness → learning gain → confidence gain	Learning gain ~ X	0.354	0.123	0.006	0.107	0.600
	Y ~ learning gain	0.572	0.073	< .001	0.426	0.717
	Total effect	0.321	0.096	0.002	0.128	0.514
	Direct effect	0.135	0.077	0.085	-0.019	0.290
	Indirect effect	0.186	0.091	0.054	0.007	0.365
GPT usefulness → manageability → learning gain	Manageability ~ X	0.453	0.108	< .001	0.236	0.670
	Y ~ manageability	0.643	0.109	< .001	0.425	0.862
	Total effect	0.354	0.123	0.005	0.108	0.599
	Direct effect	0.081	0.119	0.500	-0.157	0.319
	Indirect effect	0.273	0.091	0.001	0.124	0.495
Manageability → learning gain → satisfaction	Learning gain ~ X	0.644	0.112	< .001	0.420	0.868
	Y ~ learning gain	0.617	0.100	< .001	0.416	0.818
	Total effect	0.599	0.114	< .001	0.370	0.827
	Direct effect	0.314	0.129	0.019	0.054	0.573
	Indirect effect	0.285	0.092	< .001	0.131	0.494
Integrated support → manageability → learning gain	Manageability ~ X	0.394	0.112	0.001	0.170	0.618
	Y ~ manageability	0.644	0.112	< .001	0.420	0.868
	Total effect	0.352	0.124	0.006	0.104	0.601
	Direct effect	0.119	0.116	0.311	-0.114	0.352
	Indirect effect	0.233	0.100	0.006	0.070	0.474
Instructor support → manageability → learning gain	Manageability ~ X	0.215	0.122	0.083	-0.029	0.459
	Y ~ manageability	0.644	0.112	< .001	0.420	0.868
	Total effect	0.220	0.131	0.099	-0.043	0.483
	Direct effect	0.086	0.111	0.443	-0.136	0.307
	Indirect effect	0.134	0.092	0.077	-0.002	0.374

Table 7 Full mediation outputs for the exploratory mediation analyses. All variables were standardized before estimation. Confidence intervals are bootstrap confidence intervals based on 5,000 resamples.

E.2. Compact Path Model Outputs

The compact observed-variable path model discussed in Section 5.5 was estimated using `semopy` with maximum likelihood estimation. All variables were standardized before estimation, so all reported coefficients are standardized path estimates. The model was not intended as a definitive confirmatory structural equation model, but as a joint test of whether the proposed mechanism remains coherent when all pathways are estimated simultaneously. Given the modest

sample size ($N = 66$), the composite nature of the survey indices, and the number of estimated paths, fit statistics are treated as diagnostic evidence that the proposed path structure is compatible with the data rather than as evidence of causal identification.

The optimization converged successfully using the SLSQP algorithm in 17 iterations. Fit indices were generally favorable: $\chi^2(52) = 22.354$, $p = 1.000$, $GFI = 0.940$, $AGFI = 0.915$, $NFI = 0.940$, and $RMSEA = 0.000$. The CFI and TLI values exceeded 1.0, which can occur in small samples when the specified model substantially outperforms the baseline independence model; these values should be interpreted cautiously and are reported as returned by the estimation software.

Tables 8 and 9 report the full structural path estimates and the estimated covariances and residual variances, respectively. Table 10 reports the complete set of fit diagnostics.

Outcome	**Predictor**	**Estimate**	**SE**	z	p
Task manageability					
	GPT usefulness	0.378	0.104	3.640	< .001
	Integrated support	0.236	0.103	2.287	0.022
	OR proficiency	-0.025	0.104	-0.244	0.807
	Programming proficiency	-0.017	0.098	-0.173	0.863
	LLM proficiency	0.137	0.115	1.199	0.230
	Verification tendency	0.197	0.109	1.818	0.069
Learning gain					
	Task manageability	0.504	0.117	4.323	< .001
	GPT usefulness	0.108	0.117	0.919	0.358
	Difficulty	0.030	0.103	0.289	0.773
	OR proficiency	-0.036	0.105	-0.339	0.735
	Programming proficiency	0.151	0.102	1.472	0.141
Confidence gain					
	Learning gain	0.513	0.072	7.154	< .001
	Baseline confidence	-0.663	0.068	-9.814	< .001
	GPT usefulness	0.156	0.071	2.201	0.028
Satisfaction					
	Learning gain	0.422	0.116	3.641	< .001
	Task manageability	0.296	0.108	2.742	0.006
	Confidence gain	0.091	0.104	0.873	0.382
	Difficulty	-0.024	0.090	-0.264	0.792

Table 8 Structural path estimates for the compact path model. All variables were standardized before estimation. Outcomes are grouped by dependent variable.

Variable 1	**Variable 2**	**Estimate**	**SE**	z	p
Estimated covariances					
GPT usefulness	Integrated support	0.335	0.097	3.450	0.001
OR proficiency	Programming proficiency	0.122	0.037	3.315	0.001
LLM proficiency	Verification tendency	0.453	0.077	5.895	< .001
Difficulty	Task manageability	0.100	0.090	1.112	0.266
Residual variances					
Confidence gain		0.288	0.050	5.745	< .001
Learning gain		0.659	0.115	5.745	< .001
Task manageability		0.609	0.106	5.746	< .001
Satisfaction		0.520	0.091	5.745	< .001

Table 9 Estimated covariances and residual variances in the compact path model.

Fit index	**Value**
Estimation	
Sample size	66
Objective function	MLW
Optimization method	SLSQP
Convergence	Successful
Number of iterations	17
Objective value	0.339
Model fit	
Degrees of freedom	52
Baseline degrees of freedom	74
χ^2	22.354
χ^2 p-value	1.000
Baseline χ^2	373.154
GFI	0.940
AGFI	0.915
NFI	0.940
RMSEA	0.000
CFI	1.099†
TLI	1.141†
Information criteria	
AIC	51.323
BIC	108.254
Log-likelihood	0.339

† Values above 1.0 are reported as returned by the estimation software and should be interpreted cautiously (see text).

Table 10 Fit diagnostics for the compact path model, organized by estimation settings, model fit indices, and information criteria.

E.3. Qualitative Coding Procedure and Supplementary Outputs

Open-ended responses to three post-survey questions were analyzed using a deductive-inductive thematic coding procedure. Initial codes were derived from the conceptual framework and the expected OR learning bottlenecks identified in Section 2, including formulation, implementation, debugging, and interpretation. Codes were refined iteratively during the review of responses and were not mutually exclusive: a single response could receive multiple codes. Table 11 reports the number of responses collected, and Table 12 presents the full codebook with code definitions.

Open-ended question	**Total**	**Non-empty**	**Coded**
What aspect helped your learning the most?	66	62	60
What was most difficult or least helpful?	66	62	61
How was the GPT companion most or least useful?	66	59	52

Table 11 Open-ended response counts by question.

Table 13 reports theme frequencies by question, and Table 14 provides representative quotes for each code. Percentages use the number of non-empty responses as the denominator. The coded respondent-level matrix, containing one row per student-question response and one binary indicator per code, is available in the supplementary materials.

Code	Theme	Definition
C00	Unclear / non-substantive	Response is empty, too vague, or not interpretable for thematic coding.
C01	Coding / Python / Pyomo / Colab difficulty	Mentions coding, Python, Pyomo, Google Colab, scripts, programming syntax, or implementation mechanics.
C02	Translation from problem or model to code	Mentions translating business text, mathematical formulation, variables, or formulas into executable code.
C03	Mathematical or model understanding	Mentions understanding the model, objective function, constraints, variables, mathematical functions, or optimization logic.
C04	Debugging and errors	Mentions bugs, errors, wrong code, solver issues, fixing mistakes, or code not working.
C05	Solution extraction and interpretation	Mentions extracting, reading, interpreting, checking, or reporting the solution, results, output, or objective.
C06	Case and context understanding	Mentions understanding the case, context, business problem, purpose, goal, or managerial setting.
C07	Instructor scaffolding and debrief	Mentions professor, teacher, instructor explanations, introductory example, training, feedback, or debrief.
C08	GPT as step-by-step tutor and dialogue	Mentions GPT or AI as a guide, question-answer partner, tutor, explainer, step-by-step support, or source of hints.
C09	GPT coding support and code generation	Mentions GPT generating, completing, translating, or supporting code writing.
C10	GPT limitations and friction	Mentions GPT mistakes, misleading guidance, frustration, limited messages, paywall, or inability to detect or correct errors.
C11	Verification and critical use	Mentions checking GPT output, verifying correctness, uncertainty about AI-generated code, or the need to rely less on AI.
C12	Manageability and reduced barrier	Mentions that support made the task easier, clearer, faster, more structured, or helped the student avoid feeling lost.
C13	Practice, application, and autonomy	Mentions learning by doing, applying concepts, concrete practice, real examples, self-directed work, or autonomy.
C14	Peer collaboration	Mentions help, discussion, or collaboration with peers.

Table 12 Qualitative codebook. Codes were derived deductively from the framework and refined inductively during response review.

Code	Theme	Overall	Helped most	Most difficult	GPT useful/not useful
C00	Unclear / non-substantive	10 (5.5%)	2 (3.2%)	1 (1.6%)	7 (11.9%)
C01	Coding / Python / Pyomo / Colab difficulty	80 (43.7%)	13 (21.0%)	40 (64.5%)	27 (45.8%)
C02	Translation from problem or model to code	14 (7.7%)	5 (8.1%)	4 (6.5%)	5 (8.5%)
C03	Mathematical or model understanding	25 (13.7%)	10 (16.1%)	8 (12.9%)	7 (11.9%)
C04	Debugging and errors	26 (14.2%)	3 (4.8%)	9 (14.5%)	14 (23.7%)
C05	Solution extraction and interpretation	18 (9.8%)	9 (14.5%)	7 (11.3%)	2 (3.4%)
C06	Case and context understanding	45 (24.6%)	18 (29.0%)	11 (17.7%)	16 (27.1%)
C07	Instructor scaffolding and debrief	15 (8.2%)	13 (21.0%)	2 (3.2%)	0 (0.0%)
C08	GPT as step-by-step tutor and dialogue	52 (28.4%)	23 (37.1%)	10 (16.1%)	19 (32.2%)
C09	GPT coding support and code generation	17 (9.3%)	3 (4.8%)	3 (4.8%)	11 (18.6%)
C10	GPT limitations and friction	23 (12.6%)	1 (1.6%)	8 (12.9%)	14 (23.7%)
C11	Verification and critical use	3 (1.6%)	1 (1.6%)	1 (1.6%)	1 (1.7%)
C12	Manageability and reduced barrier	20 (10.9%)	6 (9.7%)	4 (6.5%)	10 (16.9%)
C13	Practice, application, and autonomy	10 (5.5%)	9 (14.5%)	1 (1.6%)	0 (0.0%)
C14	Peer collaboration	2 (1.1%)	2 (3.2%)	0 (0.0%)	0 (0.0%)

Table 13 Theme frequencies by open-ended question. Themes are not mutually exclusive; percentages use non-empty responses as denominators.

Code	Theme	Question	Representative quote
C01	Coding / Python / Pyomo / Colab difficulty	Most difficult	"Understand how to code."
C01	Coding / Python / Pyomo / Colab difficulty	Most difficult	"The accuracy of coding."
C02	Translation from problem or model to code	Helped most	"To translate a real problem."
C02	Translation from problem or model to code	Most difficult	"Translate from text problem to coding/math problem."
C03	Mathematical or model understanding	Most difficult	"Maybe the constraints I think."
C03	Mathematical or model understanding	GPT useful/not useful	"To explain mathematical functions or coding part."
C04	Debugging and errors	Most difficult	"Understand the errors."
C04	Debugging and errors	Most difficult	"Find errors in the code."
C04	Debugging and errors	Helped most	"Being able to understand thanks to the AI when we made errors."
C05	Solution extraction and interpretation	Most difficult	"Understand the results."
C05	Solution extraction and interpretation	Helped most	"The extraction part. It was the concrete part."
C06	Case and context understanding	Most difficult	"Reading the 12 pages of context."
C06	Case and context understanding	GPT useful/not useful	"Understand the case and debug errors."
C07	Instructor scaffolding and debrief	Helped most	"The professor's classes."
C07	Instructor scaffolding and debrief	Helped most	"The teacher explanations."
C08	GPT as step-by-step tutor and dialogue	GPT useful/not useful	"It could explain step by step."
C08	GPT as step-by-step tutor and dialogue	Helped most	"To deal with the GPT tutor."
C09	GPT coding support and code generation	GPT useful/not useful	"Most useful: generating codes."
C09	GPT coding support and code generation	Most difficult	"The writing of the code. Despite the help of the AI I could not get the solution of the case."
C10	GPT limitations and friction	Most difficult	"The limited number of chats with ChatGPT."
C10	GPT limitations and friction	GPT useful/not useful	"It was not very cooperative with me."
C11	Verification and critical use	GPT useful/not useful	"Most useful: check our solutions. Less useful: need to ask a lot of questions."
C11	Verification and critical use	Most difficult	"Understanding if the code given by AI was actually correct or not."
C12	Manageability and reduced barrier	GPT useful/not useful	"Easy Q & A and it can give clues."
C12	Manageability and reduced barrier	Helped most	"I liked the help of your ChatGPT, it is clear."
C13	Practice, application, and autonomy	Helped most	"The extraction part. It was the concrete part."
C14	Peer collaboration	Helped most	"The GPT part and discussing with friends about the case."

Table 14 Representative quotes from the open-ended responses, organized by code. Quotes are lightly edited for capitalization and punctuation only; wording is otherwise preserved.

Appendix F: Ethical Considerations

The classroom evidence presented in this article is broadly encouraging, but it does not eliminate the ethical challenges that accompany GenAI integration in education. This appendix discusses three categories of concern that OR instructors and program designers should address explicitly: overreliance and cognitive debt, equity and access, and the irreplaceable role of the instructor.

Overreliance and cognitive debt

A central risk is that students lean too heavily on automated outputs at the expense of engaging with the underlying reasoning. While AI access can boost short-term performance, it may weaken problem-transfer skills, reduce retention, and erode metacognitive awareness (Bastani et al. 2025). Kosmyna et al. (2025) describe this as the accumulation of *cognitive debt*: outsourcing intellectual effort to AI leads over time to cognitive atrophy, diminished creativity, and a weakened sense of authorship. This risk is compounded by an *illusion of mastery*, where learners feel confident because AI-assisted tasks are effortless, even though their ability to independently reproduce or adapt solutions remains limited.

These risks are especially salient in OR, where small errors in indices, constraints, or parameter definitions can produce solutions that appear reasonable but are mathematically invalid. A student who copies a generated formulation without verifying it may pass a task while learning very little. The design implication is direct: GenAI tools in OR education must include explicit verification requirements, instructor-led critique of AI outputs, and assessments that require students to demonstrate independent reasoning. The intervention evaluated in this article attempted to address this through the companion's behavioral instructions, the feasibility checker, and the instructor debrief, but longer-term studies are needed to assess whether these safeguards are sufficient to prevent cognitive debt from accumulating across a full course.

Equity and access

GenAI integration may widen existing inequalities rather than reduce them. Evidence from low- and middle-income countries shows that well-resourced schools disproportionately benefit from AI tools while under-resourced institutions risk being left further behind (De Simone et al. 2025, Khan et al. 2024). Inequities also arise within classrooms: students differ in their fluency with AI interfaces, their access to paid platforms, and their ability to critically evaluate generated outputs. Instructors similarly vary in their preparedness to design and govern AI-supported activities. These disparities risk producing uneven learning outcomes within the same course.

Further concerns arise from cultural misalignment. Most generative models are trained predominantly on data from high-income, English-language contexts, which may reduce their relevance and reliability for students working in other languages or with domain examples drawn from different regional or industrial settings. Instructors should be aware of these limitations when curating knowledge bases and designing companion prompts. Viable open-source models, shared teaching resources, and institution-sponsored platform access are important levers for making GenAI-supported OR education equitable rather than exclusionary.

The irreplaceable role of the instructor

Greater personalization through AI does not mean that students can or should learn entirely on their own. Evidence from MOOCs shows that lack of structure and mentorship fuels isolation and confusion, contributing to persistently low completion rates (Huang et al. 2023, Narayanasamy and Elçi 2020). OR is a domain that demands abstraction, judgment, and critical reasoning under uncertainty: these cannot be developed through AI dialogue alone. Instructors remain essential for structuring the learning environment, validating reasoning, managing productive struggle, and leading the reflective debriefs that consolidate understanding.

The risk of teacher displacement is therefore not merely a labor concern but a pedagogical one. If AI adoption reduces instructor involvement under the assumption that the technology can substitute for teaching, learning outcomes

will suffer. Educators must be included in adoption processes as designers and validators rather than treated as optional components of a technology-driven system. The goal is an integrated learning environment in which GenAI amplifies the instructor's reach and reduces repetitive support demands, while the instructor remains the architect of the learning experience.

References

Artigues C, Demassey S, Néron E (2008) Resource constrained project scheduling: Models, algorithms, extensions and applications. control systems. *Robotics and Manufacturing Series. ISTE-WILEY* .

Bastani H, Bastani O, Sungu A, Ge H, Kabakcı Ö, Mariman R (2025) Generative ai without guardrails can harm learning: Evidence from high school mathematics. *Proceedings of the National Academy of Sciences* 122(26):e2422633122.

De Simone ME, Tiberti FH, Rodriguez MRB, Manolio FA, Mosuro W, Dikoru EJ (2025) From chalkboards to chatbots: Evaluating the impact of generative ai on learning outcomes in nigeria. Technical report, The World Bank.

Demeulemeester EL, Herroelen WS (2002) *Project scheduling: a research handbook* (Springer).

Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, Wang L, Chen W, et al. (2022) Lora: Low-rank adaptation of large language models. *ICLR* 1(2):3.

Huang H, Jew L, Qi D (2023) Take a mooc and then drop: A systematic review of mooc engagement pattern and dropout factor. *Heliyon* 9(4).

Khan MS, Umer H, Faruqe F (2024) Artificial intelligence for low income countries. *Humanities and Social Sciences Communications* 11(1):1–13.

Kosmyna N, Hauptmann E, Yuan YT, Situ J, Liao XH, Beresnitzky AV, Braunstein I, Maes P (2025) Your brain on chatgpt: Accumulation of cognitive debt when using an ai assistant for essay writing task. *arXiv preprint arXiv:2506.08872* .

Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih Wt, Rocktäschel T, et al. (2020) Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems* 33:9459–9474.

Narayanasamy SK, Elçi A (2020) An effective prediction model for online course dropout rate. *International Journal of Distance Education Technologies (IJDET)* 18(4):94–110.

Pardo EG, Gil-Borrás S, Alonso-Ayuso A, Duarte A (2024) Order batching problems: Taxonomy and literature review. *European Journal of Operational Research* 313(1):1–24.

PÉREZ ARMAS LF (2024) Resource planning at l'audobarquoise of saint-omer: A graph theory and mathematical programming case study with python. Technical Report 624-0030-4, The Case Centre, version: 22-Jul-2024.

Sánchez JMG (2022) Building and solving mathematical programming models. *International Series in Operations Research and Management Science* .

Schiwindt C, Zimmermann J (2015) *Handbook on Project Management and Scheduling vol. 1* (Springer International Publishing).

Schwindt C, Zimmermann J, eds. (2015) *Handbook on Project Management and Scheduling*, volume 2 of *International Handbooks on Information Systems* (Springer), ISBN 978-3-319-05915-0, URL https://link.springer.com/book/10.1007/978-3-319-05915-0.

Singh A, Ehtesham A, Kumar S, Khoei TT (2025) Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136* .

Vanhoucke M, et al. (2012) *Project management with dynamic scheduling* (Springer).

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. (2017) Attention is all you need [j]. *Advances in neural information processing systems* 30(1):261–272.

Williams HP (2013) *Model building in mathematical programming* (John Wiley & Sons).