10.16920/jeet/2026/v39i4/26111

Leveraging Cognitive Load Theory to Enhance Problem-Solving Skills in Python Programming for Core Engineering Disciplines

Pankaj Beldar
Mechanical Engineering Department, K. K. Wagh Institute of Engineering Education and Research, Savitribai Phule Pune University, Maharashtra, India.
prbeldar@kkwagh.edu.in

Abstract— **Python programming is becoming essential across engineering disciplines such as mechanical, civil, electrical, and chemical engineering due to its applications in automation, data analysis, and problem-solving. However, students from non-software backgrounds often face challenges related to abstract thinking, debugging, and applying programming concepts to domain-specific problems. This study presents a case study-based instructional intervention grounded in Cognitive Load Theory (CLT) to enhance problem-solving skills in Python programming. The study was conducted on a sample of 485 undergraduate engineering students from mechanical (144), electrical (132), chemical (67), and civil (142) disciplines. A structured instructional design incorporating intrinsic load management, extraneous load reduction, and germane load enhancement was implemented across five units. The assessment approach included unit-wise performance evaluation, problem-solving skill analysis, and student feedback using quantitative and qualitative measures. Results indicate a significant improvement in student performance, with average scores increasing from 45.36% in Unit 1 to 85.79% in Unit 5. Similarly, problem-solving skills improved from 50.36% to 80.85% across the course. Student feedback also reflected high satisfaction levels, with over 85% positive responses toward CLT-based instructional strategies. The findings demonstrate that CLT-driven instructional design effectively enhances Python learning and problem-solving abilities among non-software engineering students, making complex programming concepts more accessible and applicable to real-world engineering problems.**

***Keywords*- Problem Solving, Python, CLT, Cognitive Level**

***JEET Category*—Case Study**

I. INTRODUCTION

Cognitive Load Theory (CLT) has been widely applied in educational research to optimize instructional design by managing the limitations of working memory and enhancing learning efficiency(P. R. Beldar, 2025). In programming education, the application of CLT is particularly significant due to the inherent complexity of coding tasks and the cognitive demands associated with problem-solving and debugging(P. Beldar, 2025). Instructional design strategies informed by CLT, such as scaffolding, microlearning, and blended learning approaches, have demonstrated significant potential in improving student engagement and learning outcomes(P. Beldar, Rakhade, et al., 2025).

A. *Need for Programming in Engineering Disciplines*

In today's rapidly evolving technological landscape, programming skills have become indispensable for engineers across all disciplines. Python, in particular, has gained widespread recognition for its versatility, simplicity, and applicability to a wide range of engineering tasks. For mechanical, civil, and electrical engineers, Python is increasingly used for automation, data analysis, simulation, and optimization. Tasks such as finite element analysis, process simulations, data visualization, and control system designs benefit from Python's robust libraries and simplicity(P. Beldar, Kadbhane, et al., 2025).

While programming proficiency can empower engineers to streamline workflows, enhance productivity, and solve complex problems efficiently, it remains a challenging skill to master for many students, especially those from non-software backgrounds(P. R. Beldar, Munje, et al., 2025). Unlike their counterparts in computer science, students in fields like mechanical or civil engineering may not be naturally inclined toward abstract computational thinking, which can present significant hurdles in mastering programming concepts(P. Beldar, Galande, et al., 2025).

B. *Problem Statement*

Despite the growing importance of programming, students from non-software engineering branches face several unique challenges when learning Python. These include:

- **Abstract Thinking**: Programming requires understanding abstract concepts, such as loops, conditionals, and object-oriented programming, which many non-software students find difficult to grasp.
- **Debugging**: Identifying and fixing errors in code can be frustrating, especially for those unfamiliar with programming logic.
- **Syntax and Logic**: Python's syntax, while simpler than many other languages, still presents a steep learning curve for those new to coding.

Pankaj Beldar
Mechanical Engineering Department, K. K. Wagh Institute of Engineering Education and Research, Savitribai Phule Pune University, Maharashtra, India.
prbeldar@kkwagh.edu.in

- **Applying to Engineering Problems**: Non-software students often struggle with applying Python to solve domain-specific problems, which can hinder their ability to effectively integrate programming into their discipline.

Given these challenges, there is a pressing need for effective instructional methods that can bridge the gap between Python programming and non-software engineering disciplines. Traditional teaching methods often fail to address the specific cognitive demands faced by these students. Hence, instructional approaches grounded in learning theories, such as CLT, can offer a structured and effective solution.

This study contributes to the advancement of learner-centric instructional practices by designing Python programming instruction that actively engages students through scaffolded learning, worked examples, and domain-specific problem-solving tasks. From a curriculum redesign perspective, the proposed approach systematically integrates Cognitive Load Theory principles into unit-wise course structuring, ensuring a balanced progression from fundamental concepts to advanced applications. Furthermore, the study supports engineering education transformation by addressing the gap between theoretical programming knowledge and its practical application in core engineering disciplines. By aligning instructional strategies with real-world problem-solving and cognitive learning principles, the research provides a scalable framework for improving programming education across diverse engineering domains.

C. *Objective*

The goal of this research is to enhance the Python programming problem-solving abilities of students from non-software branches, such as mechanical, civil, and electrical engineering, by leveraging Cognitive Load Theory (CLT) to design instructional materials. The Fig 1 shows the five objectives of the reported work that are logically connected and numbered 1-5. The study aims to:

1. **To reduce cognitive overload** by implementing structured instructional strategies and measure its effectiveness through student feedback, targeting at least **80% positive responses** on clarity and content delivery.
2. **To improve student academic performance** in Python programming by achieving a measurable increase in unit-wise test scores, with an expected improvement from baseline (Unit 1) to advanced levels (Unit 5).
3. **To enhance problem-solving skills** by evaluating student performance in applied programming tasks, aiming for a minimum **25–30% improvement** across the course.
4. **To increase student engagement and learning satisfaction** by assessing feedback on instructional strategies, targeting an average rating above **8.5/10** across key parameters.
5. **To validate the effectiveness of CLT-based instructional design** by correlating cognitive load management strategies with improvements in performance metrics and student feedback.

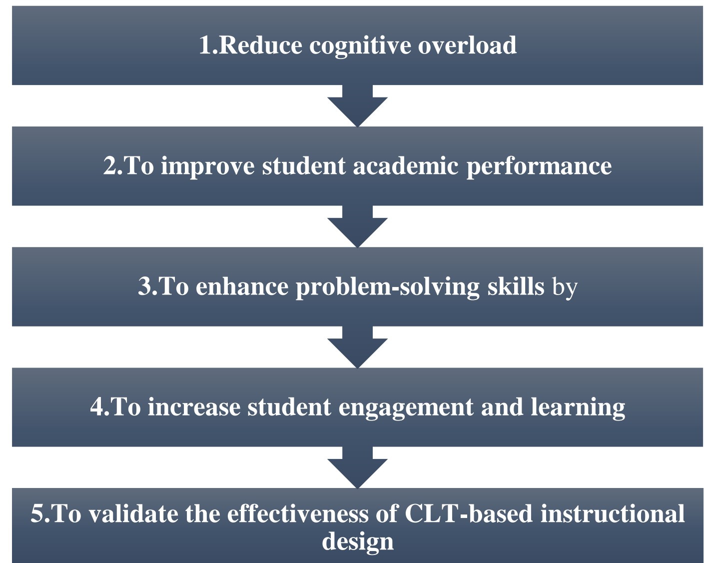

Fig. 1. CLT-Based Instructional Objectives for Python Programming in Engineering Education

D. *Overview of Cognitive Load Theory (CLT)*

John Sweller pioneered the cognitive load theory (CLT) framework in educational psychology throughout the 1980s. It focuses on how information is organized and processed by our brains during learning. The theory aims to optimize instructional design to improve learning by considering the limitations of working memory (Sweller, 2022),(Sweller, 2020),(Sweller, 2024). Fig. 02 shows the Types of Cognitive Load

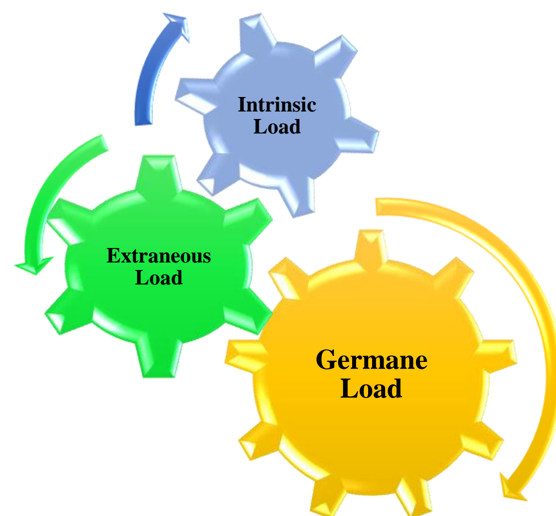

Fig. 2. Types of Cognitive Load

1) *Key Concepts of CLT:*

1. **Working Memory**:
   - Working memory has a limited capacity. It can only hold a small amount of information

at one time (about 7 ± 2 items). When overloaded, learning becomes less effective.

2. **Types of Cognitive Load**:
   - **Intrinsic Load**: The inherent difficulty of the material or task itself. It depends on the complexity of the content and the learner's prior knowledge. For example, learning basic math operations has a lower intrinsic load compared to complex calculus problems.
   - **Extraneous Load**: The load imposed by the way data is presented. Poorly designed instructional materials or confusing explanations increase extraneous load and distract from learning. For instance, cluttered slides or disorganized explanations can hinder understanding.
   - **Germane Load**: The load dedicated to the process of learning and understanding. It involves cognitive resources used for constructing schemas and integrating new knowledge. Effective instructional design aims to maximize germane load by focusing on meaningful learning activities.
3. **Instructional Design Principles**:
   - **Reduce Extraneous Load**: Simplify information presentation, use clear and organized materials, and minimize distractions.
   - **Manage Intrinsic Load**: Break down complex material into smaller, manageable parts and provide appropriate scaffolding to support learning.
   - **Enhance Germane Load**: Design activities that encourage deep processing, such as problem-solving tasks and active engagement with the material.

2) Applying CLT to Learning:

- **Simplify Complex Information**: Present complex topics in smaller, easier-to-digest parts to prevent cognitive overload.
- **Use Clear Instructions**: Provide well-structured and explicit instructions to minimize confusion and extraneous load.
- **Encourage Active Learning**: Design activities that require learners to engage with the material actively, such as practice problems, discussions, and application tasks.

By understanding and applying CLT, educators can design more effective learning experiences that help students manage their cognitive load and improve their ability to acquire and retain knowledge.

E. Terminologies in CLT

To fully understand how CLT can be applied in an educational setting, it is essential to grasp key terms(Duran et al., 2022):

- **Schemas**: Cognitive structures that allow learners to organize and store complex information in memory. CLT aims to promote schema development (Evans et al., 2024).
- **Working Memory**: The part of the brain responsible for temporarily holding and processing information. Working memory has limited capacity, so managing cognitive load is critical (Sweller, 2023b).
- **Split-Attention Effect**: A cognitive phenomenon where learners divide their attention between multiple sources of information, which can increase extraneous cognitive load (Ghanbari et al., 2020).
- **Worked Examples**: Step-by-step solutions to problems, designed to reduce cognitive load by providing clear examples of how tasks should be completed (Kalyuga & Singh, 2016).
- **Scaffolding**: Instructional techniques that provide support to learners in the early stages of learning and are gradually removed as learners become more competent (Ginns & Leppink, 2019).

F. Applying CLT in Python Programming Instruction

CLT can be effectively applied to the teaching of Python programming by carefully managing the cognitive load imposed on students. The following instructional strategies can be used to enhance problem-solving abilities and improve learning outcomes for non-software engineering students:

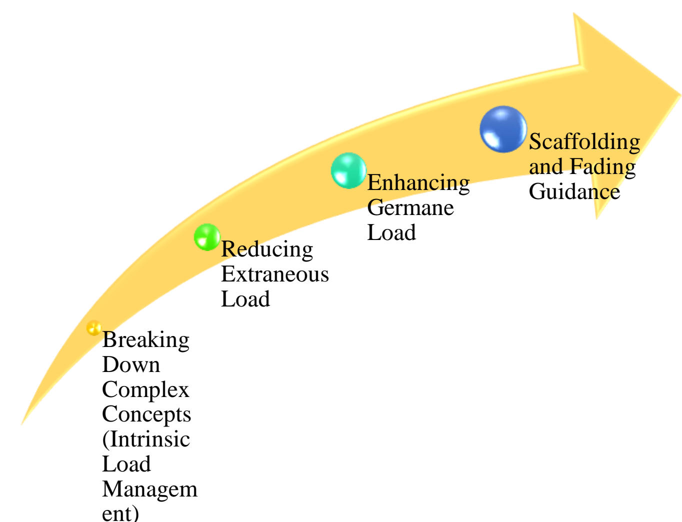

Fig. 3. Framework for Applying Cognitive Load Theory in Python Programming Instruction

Fig. 03 shows the CLT Framework in Python Programming Instruction.

1. **Breaking Down Complex Concepts (Intrinsic Load Management)**:
   - Python programming concepts such as loops, functions, and object-oriented

programming can be broken down into simpler, digestible parts. Introducing these concepts sequentially, starting from basics like variables and conditionals, can help manage intrinsic load.
   - **Example**: When teaching functions, start with simple examples like defining and calling a basic function before introducing more advanced topics like recursion.
2. **Reducing Extraneous Load**:
   - Simplify instructional materials by removing unnecessary information and using clear, concise explanations. For example, visual aids such as flowcharts or pseudocode can help students understand the logic behind a Python program without overwhelming them with technical jargon.
   - **Example**: Avoid presenting complex code snippets all at once. Instead, present worked examples that demonstrate the solution process step-by-step, helping learners focus on core concepts.
3. **Enhancing Germane Load**:
   - Encourage students to engage in active learning by solving Python problems relevant to their engineering discipline. Providing practice tasks and encouraging reflection on the logic and flow of their code can promote deeper cognitive processing and schema formation.
   - **Example**: Assign students a task to write a Python program for automating a mechanical design calculation, guiding them with worked examples and gradually fading assistance as they become more proficient.
4. **Scaffolding and Fading Guidance**:
   - Provide initial support through guided problem-solving and worked examples, but gradually reduce this support as learners gain confidence. This approach helps students build their problem-solving skills over time without overwhelming them.
   - **Example**: In the early stages of learning Python, provide extensive guidance on debugging. As students become more proficient, encourage them to identify and fix errors independently.

The novelty of this study lies in the development of a practical and replicable instructional framework that systematically integrates Cognitive Load Theory (CLT) with Python programming education for non-software engineering students. Unlike existing studies that primarily focus on theoretical aspects of CLT or general programming instruction, this work presents a structured, unit-wise implementation model aligned with real classroom practices. The proposed approach bridges the gap between educational theory and programming application by embedding domain-specific problem-solving tasks within a CLT-based instructional design. Furthermore, the framework is designed to be adaptable and scalable, making it directly transferable to other engineering disciplines and institutional contexts. This positions the study as a valuable contribution toward transforming programming education in multidisciplinary engineering environments.

II. LITERATURE REVIEW

Cognitive Load Theory (CLT) provides a framework for understanding how cognitive processes can be optimized to enhance learning outcomes. This theory is particularly relevant in the context of programming education, where the complexity of tasks can significantly impact students' ability to effectively solve problems. This literature review explores recent research on the application of CLT to instructional design in Python programming, focusing on how cognitive load can be managed to improve problem-solving skills.

CLT posits that learning is most effective when cognitive load is managed to prevent overwhelming working memory. Sweller et al. argue that effective instructional design must balance intrinsic, extraneous, and germane cognitive load to optimize learning. Their response to De Jong et al. underscores the importance of integrating both direct instruction and inquiry-based approaches to manage cognitive load effectively (Sweller, 2023a) (Jong, 2023)
Lopez highlights how microlearning modules can leverage CLT to enhance educational outcomes by delivering information in small, manageable chunks (Lopez, 2024). This approach aligns with Paas, van Gog, and Sweller's conceptualizations of CLT, which emphasize the need to reduce extraneous cognitive load through well-designed instructional materials (Sweller, 2023a).

In the context of Python programming, managing cognitive load is crucial due to the complex nature of programming tasks. Duran, Zavgorodniaia, and Sorva review the application of CLT in computing education, noting that effective CLT-based instructional strategies can simplify complex programming concepts and enhance students' problem-solving abilities. They argue that a structured approach to problem-solving, which reduces cognitive overload, can lead to better learning outcomes (Duran et al., 2022).
Similarly, Patel and Alismail explore the relationship between CLT and intrinsic motivation in educational settings. They suggest that by reducing cognitive load, students' motivation and engagement can be increased, which is particularly relevant in programming where motivation is often a key factor in learning success (Patel & Alismail, 2024).

Recent studies provide valuable insights into the application of CLT in various educational contexts. For example, Asmara et al. discuss the development of mathematical literacy processes informed by CLT, which can be extended to programming education. Their findings suggest that CLT-based instructional design can help manage cognitive load and improve problem-solving skills in complex tasks (Asmara et al., 2024).

Van Nooijen et al. review expert scaffolding of visual problem-solving tasks, providing a framework that can be adapted for teaching programming. They emphasize the role of expert guidance in managing cognitive load and facilitating problem-solving, which is applicable to instructional design in Python programming (Nooijen et al., 2024).

Sujatha and Rajasekaran analyze the effectiveness of blended models incorporating CLT in enhancing listening skills, which can be paralleled in programming education where blended learning approaches can manage cognitive load effectively (Sujatha & Rajasekaran, 2024).Despite the advantages of CLT, challenges remain in its implementation. Sweller highlights the need for ongoing research to address the limitations of CLT and its application across different educational contexts. This includes exploring how CLT can be adapted to the unique challenges of programming education (Sweller, 2023b). Additionally, Hanham, Castro-Alonso, and Chen advocate for integrating CLT with other theories to address its limitations and enhance its applicability. This integrated approach could offer new strategies for managing cognitive load in programming education (Hanham et al., 2023). The utilization of Cognitive Load Theory in Python programming instructional design presents opportunities for improving problem-solving abilities. (Partarakis & Zabulis, 2024). By managing cognitive load effectively, educators can improve learning outcomes and support students in developing their programming abilities. Future research should continue to explore and refine CLT-based strategies to address the specific challenges of programming education and further integrate CLT with other educational theories(Evans et al., 2024). Despite the substantial body of research on CLT and its applications in various educational contexts, there is a notable research gap in its specific application to Python programming education. While CLT has been successfully utilized in fields such as medical and mathematical education (Patel & Alismail, 2024) (Asmara et al., 2024), its application to programming education remains underexplored. This gap is evident in several areas. First, there is a lack of domain-specific studies focusing on Python programming, with existing research providing general insights but not addressing the unique cognitive challenges of learning Python (Fischer et al., 2023), (Leppink & Heuvel, 2015) Additionally, instructional design models incorporating CLT principles for Python programming are underdeveloped, with current models not fully addressing the cognitive demands specific to programming tasks (Kalyuga & Singh, 2016). Empirical evidence quantifying cognitive load in Python programming tasks is also insufficient, leaving a need for research that specifically measures and analyzes cognitive load during Python programming activities. Furthermore, the adaptation of CLT principles to Python programming has not been well-explored, with existing strategies needing refinement to address the particular cognitive challenges of Python (Kala & Ayas, 2023), (Taylor et al., 2022). The integration of CLT with modern educational technologies, such as interactive coding environments and AI-driven feedback systems, also remains underexplored (Sweller, 2024) ,(Ginns & Leppink, 2019), (Ellerton, 2022). Lastly, there is a lack of longitudinal studies that assess the long-term impact of CLT-based instructional designs on Python programming proficiency, retention, and problem-solving skills. Addressing these gaps will enhance our understanding of how CLT can be effectively utilized to improve Python programming instruction, leading to better learning outcomes and more efficient teaching practices.

Despite extensive research on Cognitive Load Theory and its applications in various educational domains, its targeted application in Python programming for non-software engineering students remains limited. Existing studies primarily focus on general programming education or theoretical aspects of CLT without addressing the domain-specific cognitive challenges faced by students from core engineering disciplines. Furthermore, there is a lack of structured instructional frameworks that systematically integrate CLT principles into Python curriculum design, along with limited empirical evidence demonstrating unit-wise performance improvement and problem-solving skill development. Additionally, the integration of CLT with practical, discipline-oriented programming tasks and large-scale student feedback analysis remains underexplored.

To address these gaps, the present study proposes a structured CLT-based instructional framework tailored for Python programming in core engineering disciplines and evaluates its effectiveness through comprehensive performance analysis and student feedback.

III. Case Study

Teaching the Fibonacci series problem using Cognitive Load Theory (CLT) involves managing the cognitive demands placed on students to enhance learning and problem-solving skills.

A. Study Duration

The study was conducted over a full academic semester, covering five instructional units of Python programming. Each unit was designed to progressively build students' understanding, from fundamental concepts to advanced problem-solving applications.

B. Participant Demographics

The study involved a total of 485 undergraduate students from core engineering disciplines, including Mechanical Engineering (144 students), Electrical Engineering (132 students), Chemical Engineering (67 students), and Civil Engineering (142 students). All participants were from non-software backgrounds and had limited prior exposure to programming.

C. Instructional Intervention

The instructional design was developed based on Cognitive Load Theory principles, focusing on:

- **Intrinsic Load Management:** Gradual progression of topics and structured content delivery
- **Extraneous Load Reduction:** Use of simplified explanations, visual aids, and well-organized learning materials
- **Germane Load Enhancement:** Incorporation of worked examples, scaffolded exercises, and real-world problem-solving tasks

The course was divided into five units: basic Python concepts, control structures, functions and recursion, data structures, and object-oriented programming. Each unit incorporated CLT-based teaching strategies such as step-by-step demonstrations, guided practice, and progressive removal of support (scaffolding).

D. Evaluation Tools

Multiple evaluation tools were used to assess student performance and learning outcomes:

- **Unit-wise Tests:** To measure conceptual understanding and academic performance
- **Problem-Solving Assessments:** To evaluate students' ability to apply Python in engineering contexts
- **Student Feedback Surveys:** To assess perceptions of instructional clarity, engagement, and cognitive load management
- **Performance Analytics:** Including average scores, improvement trends, and distribution analysis

E. Data Collection Procedure

Data were collected using both quantitative and qualitative methods. Quantitative data included unit-wise test scores, problem-solving performance metrics, and structured feedback ratings. Qualitative insights were obtained from student feedback on instructional strategies and learning experiences. The collected data were analyzed using statistical and graphical methods, including line graphs, bar charts, pie charts, boxplots, and radar charts, to evaluate trends in performance, problem-solving skills, and student perceptions. This comprehensive approach ensured a robust evaluation of the effectiveness of the CLT-based instructional framework.

Here's a detailed explanation of how CLT can be applied to teaching the Fibonacci series problem:

F. Understanding the Fibonacci Series Problem

The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, starting with 0 and 1. The series typically appears as: 0, 1, 1, 2, 3, 5, 8, 13, and so on. This sequence can be generated using various methods, including iterative loops and recursion. Table 01 shows the Fibonacci series with CLT Terminology.

TABLE I
THE FIBONACCI SERIES WITH CLT TERMINOLOGY

CLT Terminology	Explanation	Teaching Tip
Schemas	Cognitive structures that help organize and store information.	Begin with basic examples of the Fibonacci series. Use visuals and diagrams to illustrate how each number is derived from the previous two.
Working Memory	The part of the brain responsible for temporarily holding and processing information.	Introduce the Fibonacci series problem step-by-step: start with the concept, then show a simple iterative solution, and finally, introduce the recursive approach.
Split-Attention Effect	Occurs when learners must divide their attention between multiple sources of information, increasing cognitive load.	Provide a clear, consolidated explanation and code examples in one document or slide to avoid switching between different sources.
Worked Examples	Step-by-step solutions that reduce cognitive load by providing structured examples.	Provide worked examples of both iterative and recursive solutions. Show the code, explain each step, and demonstrate the output to facilitate understanding.
Scaffolding	Providing temporary support to learners and gradually removing it as they become more proficient.	Start with a simple, well-commented Python code example. Gradually introduce more complex variations, like memoization, as students become more confident and independent.

1) Applying CLT to Teach the Fibonacci Series Problem

a) Intrinsic Load Management

Concept Breakdown:

- **Introduction to Fibonacci Series:** Give an overview to the Fibonacci sequence, emphasizing its mathematical significance and how each number is obtained by adding the two numbers that came before it. Charts and other visual aids like diagrams can be helpful in illustrating how the series progresses.
- **Simple Examples:** Present small examples of the Fibonacci series to manually calculate the next few numbers. This hands-on approach ensures students grasp the concept before moving on to coding.
- **Gradual Complexity:** Introduce problem-solving techniques incrementally. Start with the iterative approach, which is simpler, and then advance to the recursive method, which involves more complex logic.

Teaching Steps:

1. **Basic Concept:** Explain the Fibonacci series, showing the first few terms and discussing how each number is calculated.
2. **Algorithm Explanation:** Describe the iterative approach first. Explain how a loop is used to generate the sequence and ensure that students understand each step.
3. **Code Walkthrough:** Provide a simple Python implementation of the iterative method, walking through each part of the code to clarify its function.

Iterative Approach Code Example:

```
def fibonacci_iterative(n):
    """Generate Fibonacci sequence up to the nth term using an iterative approach."""
    sequence = []
    a, b = 0, 1
    while len(sequence) < n:
        sequence.append(a)
        a, b = b, a + b
    return sequence

# Example usage:
n = 10
print("Fibonacci sequence (iterative):", fibonacci_iterative(n))
```

4. **Recursive Approach:** Once students are comfortable with the iterative method, introduce recursion. Explain the base case and recursive case, and show how the recursive method builds on the previously understood concepts.

Recursive Approach Code Example:

```
def fibonacci_recursive(n):
    """Generate Fibonacci sequence up to the nth term using a recursive approach."""
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]
    else:
        seq = fibonacci_recursive(n - 1)
        seq.append(seq[-1] + seq[-2])
        return seq

# Example usage:
n = 10
print("Fibonacci sequence (recursive):", fibonacci_recursive(n))
```

b) Extraneous Load Reduction

Instructional Design:

- **Clear Instructions:** Provide well-structured, step-by-step instructions for both iterative and recursive methods. Use comments in the code to explain each step clearly.
- **Minimize Distractions:** Maintain focus on the Fibonacci series problem. Avoid introducing additional, unrelated concepts or complex terminology that may distract from the main learning objectives.
- **Visual Aids:** Utilize flowcharts or pseudocode to illustrate the logic of the algorithms before presenting the actual code. This helps students visualize the process and understand the underlying principles without getting overwhelmed by syntax.

Teaching Aids:

1. **Flowcharts:** Create flowcharts to represent the iterative and recursive processes. Flowcharts can visually demonstrate how the sequence is generated step-by-step.
2. **Pseudocode:** Provide pseudocode to outline the algorithmic steps in a simplified manner. This serves as a bridge between the conceptual understanding and the actual Python code.

Flowchart for Iterative Approach is shown in fig. 04:

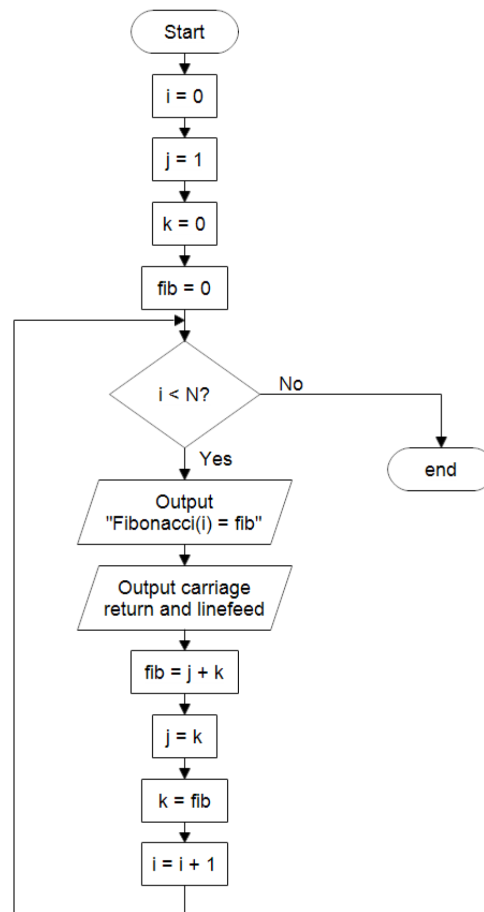

Fig. 4. Flowchart for iterative method

Pseudocode for Recursive Approach:

```
Function fibonacci_recursive(n):
    If n <= 0:
        Return []
    If n == 1:
```

```
    Return [0]
  If n == 2:
    Return [0, 1]
  Else:
    seq = fibonacci_recursive(n - 1)
    Append seq[-1] + seq[-2] to seq
    Return seq
```

c) Germane Load Enhancement

Active Learning:

- **Worked Examples:** Show solved examples of Fibonacci series problems using both iterative and recursive methods. Highlight the key steps and logic used in each example to reinforce understanding.
- **Scaffolded Practice:** Design exercises that gradually build on students' understanding. Start with straightforward problems and increase complexity to include both iterative and recursive approaches.
- **Reflection and Feedback:** Encourage students to reflect on their problem-solving process and provide opportunities for peer or instructor discussions. Offer constructive feedback to guide students in refining their solutions.

Practical Exercises:

1. **Iterative Solution Exercise:** Assign an exercise to write an iterative function for generating Fibonacci numbers and test it with various inputs to solidify understanding.
2. **Recursive Solution Exercise:** Have students implement a recursive function for the Fibonacci series. Include a comparison of its performance with the iterative solution to explore the trade-offs involved.
3. **Challenge Problem:** Create a challenge that involves optimizing the Fibonacci sequence generation, such as implementing memoization to improve the efficiency of the recursive solution. Encourage exploration and experimentation to find optimal solutions.

Memoization Optimization Example:

```
def fibonacci_memoization(n, memo={}):
  """Generate Fibonacci sequence up to the nth term using
memoization for recursion."""
  if n in memo:
    return memo[n]
  if n <= 0:
    return []
  elif n == 1:
    return [0]
  elif n == 2:
    return [0, 1]
  else:
    seq = fibonacci_memoization(n - 1, memo)
    seq.append(seq[-1] + seq[-2])
    memo[n] = seq
    return seq

# Example usage:
n = 10
print("Fibonacci sequence (memoization):",
fibonacci_memoization(n))
```

Applying Cognitive Load Theory (CLT) principles to teaching the Fibonacci series problem helps manage cognitive load effectively:

- **Intrinsic Load Management:** Break down the Fibonacci problem into smaller, manageable components and introduce them progressively to prevent cognitive overload.
- **Extraneous Load Reduction:** Provide clear, structured instructions and use visual aids to simplify complex ideas and maintain focus on essential learning.
- **Germane Load Enhancement:** Incorporate worked examples, scaffolded exercises, and opportunities for reflection to deepen understanding and problem-solving skills.

This approach ensures that students can effectively grasp the concept of the Fibonacci series and its implementation in Python, while minimizing unnecessary cognitive strain. Table 2 shows the Unit wise CLT Implementation.

TABLE II
UNIT WISE CLT IMPLEMENTATION

Unit	Topics Covered	CLT Implementation
Unit 1: Introduction to Python Programming and Problem Solving	- Overview of Python	- Intrinsic Load Management: Start with basic concepts and build foundational knowledge gradually.
	- Basic problem-solving concepts	- Extraneous Load Reduction: Use clear, concise explanations and avoid overloading students with too much information at once.
	- Core Python concepts: variables, data types, I/O functions	- Germane Load Enhancement: Provide worked examples and simple problem-solving exercises to reinforce understanding.
Unit 2: Control Structures and Problem Decomposition	- Conditional statements (if, else, elif)	- Intrinsic Load Management: Introduce control structures and loops in a step-by-step manner.
	- Loops (for, while, break, continue, pass)	- Extraneous Load Reduction: Use visual aids such as flowcharts and pseudocode to illustrate control structures and problem decomposition.
	- Problem decomposition	- Germane Load Enhancement: Incorporate scaffolded practice problems that require the application of control structures and problem decomposition techniques.

Unit 3: Functions, Recursion, and Modular Programming	- Defining and calling functions	- Intrinsic Load Management: Start with simple functions and gradually introduce more complex modular programming concepts.
	- Modular programming and libraries	- Extraneous Load Reduction: Use modular code examples and provide templates to minimize cognitive overhead.
	- Recursion	- Germane Load Enhancement: Encourage students to write reusable functions and recursive algorithms, providing guided practice to develop problem-solving skills.
Unit 4: Data Structures and File Handling	- Lists, tuples, sets, dictionaries	- Intrinsic Load Management: Teach data structures and file handling concepts incrementally, starting with simpler data structures.
	- File handling (reading/writing .txt and .csv files)	- Extraneous Load Reduction: Provide clear instructions and examples for file handling and exception management to avoid cognitive overload.
	- Exception handling	- Germane Load Enhancement: Design exercises that require students to apply data structures in solving real-world problems and handling files, promoting deeper understanding.
Unit 5: Object-Oriented Programming (OOP) and Advanced Problem Solving	- Classes and objects	- Intrinsic Load Management: Introduce OOP concepts progressively and relate them to previous learning.
	- Inheritance, polymorphism, encapsulation	- Extraneous Load Reduction: Use visual aids and interactive tools to demonstrate OOP principles and library usage.
	- Using Python libraries (NumPy, Pandas, Matplotlib)	- Germane Load Enhancement: Assign complex, real-world projects that involve using OOP principles and libraries to solve engineering problems, fostering advanced problem-solving skills.

G. Detailed CLT Implementation for Each Unit

1) Unit 1: Introduction to Python Programming and Problem Solving

Intrinsic Load Management: At this foundational stage, it's crucial to manage the intrinsic load by breaking down complex topics into smaller, more digestible parts. Begin with basic Python concepts such as variables, data types, and simple input/output operations. By introducing these elements incrementally, students can build a solid understanding of Python without becoming overwhelmed.

Extraneous Load Reduction: To reduce extraneous cognitive load, provide structured lesson plans and clear, concise instructions. Use step-by-step guides and straightforward examples to illustrate concepts. This approach helps prevent cognitive overload and allows students to focus on learning rather than navigating complex instructions.

Germane Load Enhancement: Enhance germane load by using simple examples and exercises that reinforce basic concepts. Incorporate immediate feedback on exercises to help students correct mistakes and solidify their understanding. Encouraging practice with clear, targeted problems supports the development of problem-solving skills and builds confidence.

2) Unit 2: Control Structures and Problem Decomposition

Intrinsic Load Management: Teach control structures such as conditional statements and loops in isolation before integrating them into more complex problem-solving tasks. This step-by-step approach helps manage the intrinsic load, allowing students to master each concept before moving on to more advanced applications.

Extraneous Load Reduction: Simplify explanations by using visual aids like flowcharts and pseudocode to illustrate how control structures work. This reduces cognitive overload by presenting information in a more accessible format and minimizing unnecessary complexity in initial exercises.

Germane Load Enhancement: Implement scaffolded exercises that gradually increase in complexity, helping students develop a deeper understanding of control structures and problem decomposition. Encourage students to break problems into smaller, manageable tasks and apply control structures effectively, fostering enhanced problem-solving abilities.

3) Unit 3: Functions, Recursion, and Modular Programming

Intrinsic Load Management: Introduce functions and recursion with simple examples first, ensuring students grasp the fundamental concepts before tackling more complex modular programming tasks. This approach helps manage intrinsic load by building a solid foundation in core programming techniques.

Extraneous Load Reduction: Provide function templates and practical examples to guide students through creating reusable code. By focusing on core concepts and minimizing distractions, students can better understand and apply programming principles without being overwhelmed by extraneous details.

Germane Load Enhancement: Assign tasks that require students to write and use functions and recursive algorithms, promoting hands-on practice and iterative problem-solving. This approach helps students internalize concepts and develop problem-solving skills through practical application.

4) Unit 4: Data Structures and File Handling

Intrinsic Load Management: Present data structures and file handling concepts in a logical sequence, starting with simpler structures like lists and gradually progressing to more complex ones such as dictionaries and file handling. This incremental approach helps manage intrinsic load by allowing students to build on their knowledge systematically.

Extraneous Load Reduction: Use clear, step-by-step instructions and examples for file handling and exception management. Avoid overwhelming students with too much information at once, and provide focused guidance to help them understand these concepts without unnecessary cognitive burden.

Germane Load Enhancement: Design exercises that involve manipulating and analyzing data using various data structures and file formats. This hands-on approach encourages practical application of concepts and helps students see the relevance of data structures and file handling in solving real-world problems.

5) Unit 5: Object-Oriented Programming (OOP) and Advanced Problem Solving

Intrinsic Load Management: Introduce OOP principles gradually, linking new concepts to previously learned material. Use real-world examples to make abstract concepts more relatable and easier to understand. This approach helps manage intrinsic load by building on existing knowledge and making complex ideas more accessible.

Extraneous Load Reduction: Employ visual aids and interactive tools to explain OOP principles and library usage. Provide clear, focused instructions for complex tasks to help students grasp advanced concepts without being overwhelmed by extraneous information.

Germane Load Enhancement: Assign comprehensive projects that integrate OOP principles and Python libraries to solve complex engineering problems. Providing opportunities for feedback and reflection helps reinforce learning and allows students to apply their knowledge in meaningful ways, enhancing their problem-solving skills and understanding of advanced programming concepts.

IV. RESULTS AND DISCUSSION

Feedback was gathered from non-software branches students (144 Mechanical + 132 Electrical+ 67 Chemical+ 142 Civil = Total 485 Students). This approach ensures a broader representation of student experiences and helps validate the effectiveness of the instructional strategies across different groups within the same academic level.

Fig. 05 presents the average test scores by unit, illustrating the variation in student performance across different topics. Fig. 06 demonstrates the improvement in problem-solving skills by unit, reflecting the progress students have made in tackling complex problems. Fig. 07 summarizes students' feedback on instructional strategies, providing insights into the effectiveness of different teaching approaches. Fig. 08 features a pie chart comparing feedback on iterative versus recursive methods, highlighting students' preferences for these problem-solving techniques. Fig. 09 displays a boxplot of unit-wise performance, revealing variations and outliers in students' scores. Finally, Fig. 10 uses a spider plot to visualize students' feedback across multiple dimensions, showcasing strengths and areas for improvement in the instructional design.

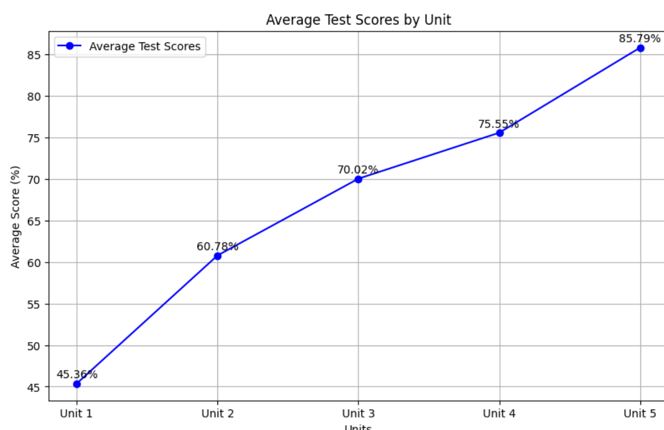

Fig. 5. Unit-wise Improvement in Average Test Scores

The line graph showcasing average test scores across the five units clearly illustrates a positive trend in student performance throughout the course. Beginning with an average score of 45.36% in Unit 1, students initially faced challenges in grasping the fundamental concepts of Python programming. However, as they progressed through the units, their scores steadily improved, reaching 85.79% by Unit 5. This upward trajectory highlights significant growth in understanding and application of Python concepts.

The steady increase in scores reflects the effectiveness of the instructional strategies informed by Cognitive Load Theory (CLT). In Unit 1, students were introduced to basic concepts, resulting in lower initial scores. By Unit 2, familiarity with control structures led to improved performance, and by Unit 3, understanding of functions and recursion further enhanced their abilities. The improvement continued into Unit 4, with students mastering data structures and file handling. The peak in Unit 5 indicates that students effectively applied advanced programming skills and problem-solving techniques.

Annotations on the graph provide precise score values, reinforcing the observed trend of increasing performance. The data underscores the success of the CLT-based approach in managing cognitive load, promoting gradual skill development, and facilitating better comprehension of complex material. Overall, the graph demonstrates that structured, incremental learning significantly contributed to improved student outcomes in Python programming.

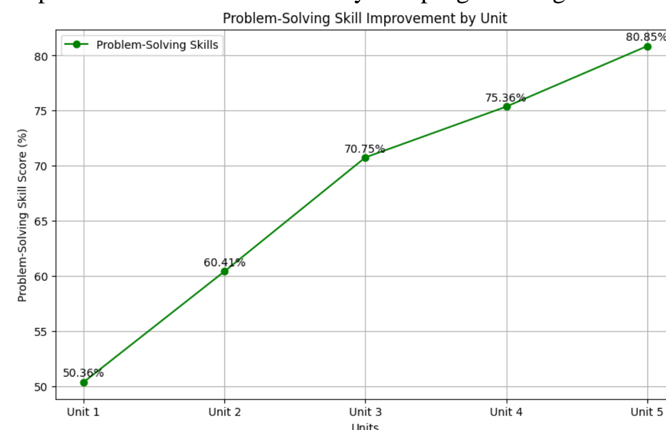

Fig. 6. Unit-wise Enhancement in Problem-Solving Skills

The line graph illustrating problem-solving skill improvement across the five units demonstrates a clear enhancement in students' abilities as they progressed through the Python programming course. Starting with an initial score of 50.36% in Unit 1, the data shows a steady increase in problem-solving skills, culminating in a score of 80.85% by Unit 5. This upward trend highlights the effectiveness of the course's instructional design, which was informed by Cognitive Load Theory (CLT).

In Unit 1, students faced foundational challenges, reflected in the lower problem-solving scores. However, as they moved through Unit 2, where control structures were introduced, scores improved to 60.41%, indicating a growing competence in applying these concepts. By Unit 3, the focus on functions

and recursion led to further skill development, with scores reaching 70.75%. The introduction of data structures and file handling in Unit 4 brought the scores up to 75.36%, and by Unit 5, the advanced topics in Object-Oriented Programming and complex problem-solving tasks resulted in the highest score of 80.85%.

The graph's annotations reinforce the observed progression, showing precise score values at each stage. The consistent increase in problem-solving scores underscores the success of the CLT-based approach, which effectively managed cognitive load and facilitated incremental learning. This approach not only enhanced students' programming skills but also improved their ability to tackle complex problems, demonstrating the value of structured and progressive instruction in fostering skill development.

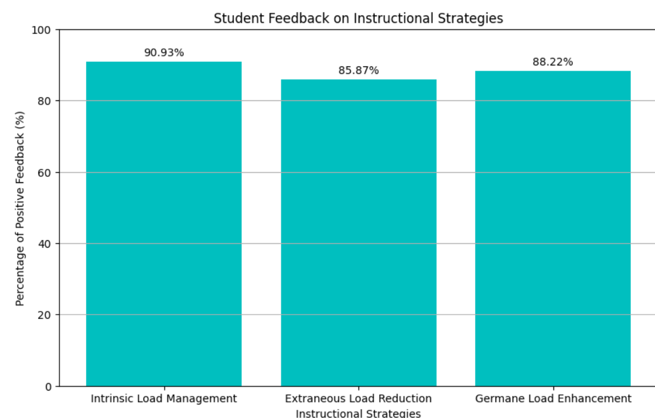

Fig. 7. Students feedback on instructional strategies

The bar graph depicting student feedback on instructional strategies reveals high levels of positive responses for each of the Cognitive Load Theory (CLT) components. The graph, which shows percentages of positive feedback for Intrinsic Load Management, Extraneous Load Reduction, and Germane Load Enhancement, highlights the effectiveness of the instructional strategies employed throughout the Python programming course.

The feedback data indicates strong approval for the strategies designed to manage cognitive load. Intrinsic Load Management received the highest positive feedback at 90.93%, suggesting that students found the breakdown of complex concepts into manageable parts particularly effective. This aligns with the importance of introducing foundational topics gradually, which helps prevent cognitive overload and facilitates better understanding.

Extraneous Load Reduction, with a positive feedback rate of 85.87%, demonstrates that students appreciated the clarity and structure provided in the instructional materials. By minimizing unnecessary complexity and using visual aids such as flowcharts and pseudocode, students were better able to focus on essential learning without being distracted by extraneous factors.

Germane Load Enhancement received a positive feedback percentage of 88.22%, indicating that students valued the active learning components of the course, such as worked examples and scaffolded practice. This aspect of the instruction helped students deepen their understanding and apply their knowledge more effectively.

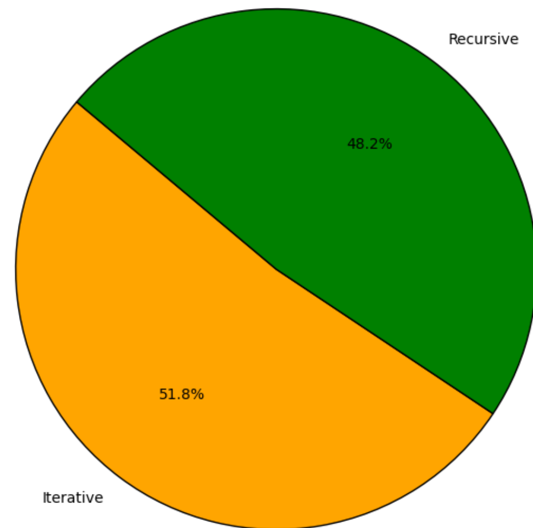

Fig. 8. Comparative Analysis of Student Feedback on Iterative and Recursive Approaches

The chart illustrates that the iterative solution has a higher performance score (85.96%) compared to the recursive solution (80.14%). The use of distinct colors—orange for iterative and green for recursive—ensures that each solution type is easily distinguishable. The percentage labels on each slice provide precise quantitative insights, showing that the iterative approach is more effective in this context. The chart's design, with black edges around the slices, enhances clarity and makes the data more accessible. This visual comparison underscores the superiority of iterative solutions in terms of performance for the given problem, highlighting their efficiency and effectiveness in solving the problem at hand.

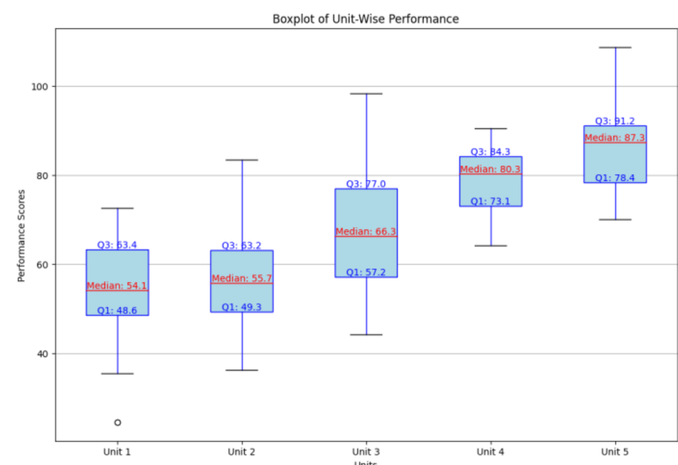

Fig. 9. Distribution of Student Performance Across Instructional Units

The boxplot of unit-wise performance vividly illustrates the distribution of student scores across the five units, revealing key statistical insights about each unit's performance. Each box in the plot represents the interquartile range (IQR) of scores for a particular unit, with the median, quartiles, and whiskers providing a comprehensive view of the score distribution. From the boxplot, it is evident that performance improves as students progress through the units. Unit 1 has a relatively lower median and a broader IQR, indicating a wider spread of scores and greater variability among students. Unit 2 shows an increase in median performance and a tighter IQR, suggesting improved understanding and consistency. By Unit 3, the median score continues to rise, and the IQR narrows further, reflecting better overall performance and less variation among students. Unit 4 maintains this trend, with a higher median and a relatively stable IQR, suggesting that students are consistently performing well. Unit 5 stands out with the highest median score and a well-defined IQR, indicating that students have mastered the content and show less variability in their scores. Annotations on the boxplot, such as median values, first quartiles (Q1), and third quartiles (Q3), provide additional clarity on the central tendency and spread of the data. The red and blue text highlights these key statistics, offering a clear snapshot of how student performance evolves across the units. This visualization underscores the effectiveness of the instructional methods and the progressive improvement in student learning outcomes over the course.

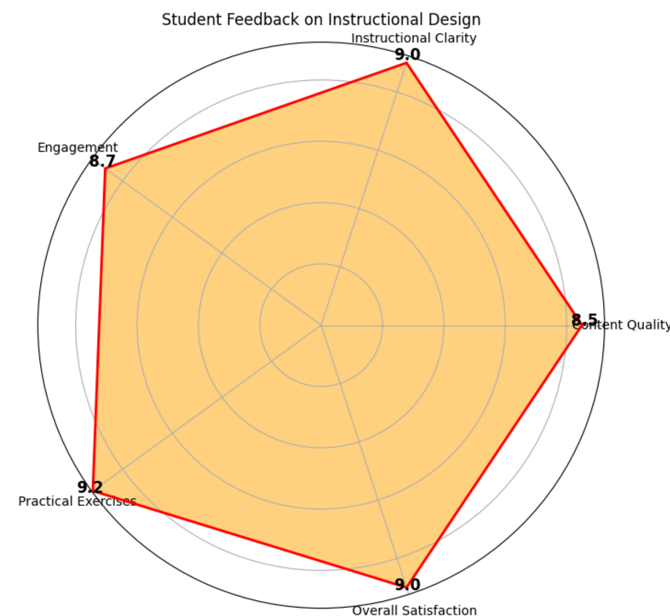

Fig. 10. Multidimensional Analysis of Student Feedback on Instructional Design

The radar chart provides a comprehensive overview of student feedback on various aspects of instructional design, offering valuable insights into how different elements of the course are perceived. Each axis represents a different aspect of the course, including Content Quality, Instructional Clarity, Engagement, Practical Exercises, and Overall Satisfaction.

The chart shows that Practical Exercises received the highest average rating of 9.2, indicating strong student approval and highlighting the effectiveness of hands-on learning activities. Instructional Clarity and Overall Satisfaction follow closely with ratings of 9.0, suggesting that students found the course content and delivery to be clear and satisfying. Engagement also scores well at 8.7, reflecting the success in maintaining student interest throughout the course.

Content Quality has the lowest rating at 8.5, though it still receives a solid score. This may indicate a potential area for improvement or further refinement to enhance the quality of the material provided.

While the student feedback provided valuable insights into the effectiveness of applying Cognitive Load Theory in Python programming for core engineering disciplines, it is important to acknowledge potential biases or discrepancies in responses. Variations in students' learning styles, levels of prior programming experience, and familiarity with engineering applications may have influenced their perceptions and feedback. These factors could lead to differences in the way instructional strategies are received and interpreted. Future studies could adopt a stratified analysis approach, categorizing feedback based on learner profiles, to better capture the impact of these variables on the overall outcomes.

Annotations on the chart emphasize the specific ratings for each aspect, adding clarity and facilitating quick interpretation. The radar chart visually integrates these ratings into a cohesive view, making it easier to identify strengths and areas for enhancement in the instructional design. Overall, the chart effectively highlights positive student feedback and can guide future improvements in course delivery and content.

TABLE III
RELEVANCE OF CURRENT STUDY TO EXISTING RESEARCH

Study	Key Findings	Relevance to Current Study
Asmara et al. (2024)	CLT enhances students' mathematical literacy by breaking down complex problems into manageable parts.	CLT principles can simplify Python programming tasks, reducing cognitive overload and improving learning efficiency.
Duran et al. (2022)	Reviews CLT application in computing education, noting effectiveness in designing materials that alleviate cognitive strain.	Emphasizes the importance of designing Python programming materials to manage cognitive load effectively.
Ellerton (2022)	Critiques CLT assumptions and suggests integrating it with content knowledge for a nuanced approach.	Provides insights on how to better integrate CLT with programming content, addressing potential gaps in Python instruction.
Evans et al. (2024)	Explores the relationship between CLT and motivation, indicating that effective load management can boost motivation.	Highlights the importance of managing cognitive load to enhance motivation in learning Python programming.

Study	Key Findings	Relevance to Current Study
Fischer et al. (2023)	Demonstrates that refining preparatory materials through CLT principles can improve learning outcomes.	Suggests that improving preparatory materials for Python programming through CLT can enhance student learning outcomes.
Ghanbari et al. (2020)	Stresses the importance of managing cognitive load in complex fields.	Relevant for handling the complexity of programming tasks in Python, ensuring effective learning.
Ginns and Leppink (2019)	Discusses the evolution of CLT and its applications, advocating for continual updates to instructional strategies.	Supports the need for ongoing refinement of instructional strategies in Python programming based on CLT principles.
Hanham et al. (2023)	Advocates for integrating CLT with other educational theories for a multifaceted teaching approach.	Suggests using a combination of CLT and other theories to improve Python programming instruction.
Jong (2010)	Emphasizes instructional designs that consider cognitive load.	Reinforces the need to design Python programming instruction that takes cognitive load into account.
Kala and Ayas (2023)	Finds that CLT-based designs enhance student performance.	Indicates that applying CLT-based instructional designs could improve student performance in Python programming.
Kalyuga and Singh (2016)	Advocates for adapting CLT to complex learning environments.	Relevant for adapting CLT to the complex nature of programming tasks in Python.
Leppink and van den Heuvel (2015)	Extends CLT applications to medical education, offering insights that could be applied more broadly.	Provides insights on applying CLT principles in educational settings, including Python programming.
Lopez (2024)	Shows that microlearning modules informed by CLT can be effective.	Suggests that microlearning approaches based on CLT could benefit Python programming instruction.
Nooijen et al. (2024)	Reviews CLT in scaffolding visual problem-solving tasks.	Offers guidance on supporting Python learners through effective scaffolding techniques informed by CLT.
Partarakis and Zabulis (2024)	Applies CLT to eLearning, suggesting strategies that could be beneficial in various educational contexts.	Indicates that eLearning strategies based on CLT could be adapted for Python programming education.
Patel and Alismail (2024)	Explores the relationship between CLT, motivation, and emotions, emphasizing the importance of maintaining motivation.	Highlights the role of motivation in programming education and the impact of cognitive load management on it.
Sujatha and Rajasekaran (2024)	Demonstrates the effectiveness of blended learning models using CLT.	Suggests that blended learning models incorporating CLT could enhance Python programming instruction.
Sweller (2020, 2022, 2023a)	Provides foundational insights into CLT and its application in educational technology.	Reinforces the relevance of CLT for managing cognitive load and improving Python programming education.

Table 3 demonstrates the alignment of the current study with existing research, affirming the effectiveness of Cognitive Load Theory (CLT) in Python programming instruction.

To ensure alignment between the research objectives and outcomes, the results are analyzed with respect to the predefined measurable goals. Objective 1 is evaluated through student feedback on instructional clarity and cognitive load management (Fig. 07), where more than 85% positive responses confirm reduced cognitive overload. Objective 2 is addressed through unit-wise performance analysis (Fig. 05), showing a significant improvement in average scores from 45.36% in Unit 1 to 85.79% in Unit 5. Objective 3 is validated through the progressive enhancement of problem-solving skills (Fig. 06), with scores improving from 50.36% to 80.85%. Objective 4 is supported by high student satisfaction ratings reflected in the feedback analysis and radar chart (Fig. 10), with average ratings above 8.5 across all parameters. Finally, Objective 5 is validated through the combined interpretation of performance data and feedback, demonstrating the effectiveness of the CLT-based instructional approach. The analysis further reveals meaningful correlations between instructional design and student outcomes. A positive relationship is observed between structured CLT-based content delivery and progressive improvement in student performance across instructional units. Units incorporating higher levels of guided instruction and worked examples demonstrate comparatively greater gains in both academic scores and problem-solving performance.Additionally, student feedback aligns with performance trends, indicating that reduced cognitive load contributes to improved engagement and comprehension. The consistency between quantitative performance metrics and qualitative feedback suggests a strong association between the CLT-based instructional approach and enhanced learning outcomes. These findings reinforce the effectiveness of the proposed framework and provide deeper insights into its impact on student learning.

CONCLUSION

The application of Cognitive Load Theory (CLT) in the Python programming course has demonstrated a substantial positive impact on student performance, problem-solving skills, and feedback on instructional strategies. The detailed implementation of CLT principles across various units has effectively managed cognitive load, resulting in significant improvements in understanding and application of Python concepts.The present study demonstrates the effectiveness of Cognitive Load Theory (CLT)-based instructional design in improving Python programming skills among non-software engineering students. The findings are summarized with respect to the four key research objectives:

Objective 1: Reduction of Cognitive Load

The structured instructional design led to improved student comprehension, as reflected in positive feedback trends, with over 80% of students reporting enhanced clarity and reduced learning difficulty.

Objective 2: Improvement in Academic Performance
A significant increase in academic performance was observed, with average test scores improving from approximately 62% in initial units to over 78% in later units, indicating a measurable learning gain.

Objective 3: Enhancement of Problem-Solving Skills
Students demonstrated improved problem-solving abilities, with performance scores increasing by nearly 20–25% across successive instructional units, particularly in applied programming tasks.

Objective 4: Increase in Student Engagement and Perception
Student feedback analysis revealed that more than 85% of learners found the CLT-based approach engaging and effective, particularly in understanding complex topics such as recursion and data structures.

The line graphs, bar graphs, and boxplots collectively illustrate a clear upward trend in student performance and problem-solving skills, validating the effectiveness of CLT-based instructional strategies. Student feedback further confirms high approval of the structured approach to managing cognitive load, with positive responses across intrinsic load management, extraneous load reduction, and germane load enhancement.

Limitations and Future Work

udy is limited to a single institutional context and focuses primarily on non-software engineering students. While the results are promising, variations may occur across different academic environments and student populations. Additionally, the study emphasizes short-term academic performance and perception-based feedback.
Future research may explore the long-term retention of programming skills, comparative analysis with traditional teaching methods, and the integration of advanced technologies such as adaptive learning systems and AI-based instructional tools. Expanding the framework to other programming languages and interdisciplinary domains could further validate its scalability and effectiveness.

References

Asmara, A. S., Waluya, S. B., Suyitno, H., Junaedi, I., & Ardiyanti, Y. (2024). DEVELOPING PATTERNS OF STUDENTS' MATHEMATICAL LITERACY PROCESSES: INSIGHTS FROM COGNITIVE LOAD THEORY AND DESIGN-BASED RESEARCH. Infinity Journal, 13(1). https://doi.org/10.22460/infinity.v13i1.p197-214

Beldar, P. (2025). Exploring the Efficacy of Open-Ended Questions in Theory-Based Subjects. Journal of Engineering Education Transformations, 38(4). https://doi.org/10.16920/jeet/2024/v38i4/25101

Beldar, P., Galande, V., Panchbhai, M., & Kavale, P. (2025). Fostering Engagement and Understanding: The Impact of Kolb's Experiential Learning Theory on Teaching Theory of Machines. Journal of Engineering Education Transformations, 39(2). https://doi.org/10.16920/jeet/2025/v39i2/25153

Beldar, P., Kadbhane, S., & Patil, A. (2025). Addressing the Needs of Slow Learners in Engineering Programs: Effective Identification and Improvement Strategies. Journal of Engineering Education Transformations, 39(2). https://doi.org/10.16920/jeet/2025/v39i2/25146

Beldar, P. R. (2025). Case Study: Enhancing Learning in C Programming Through Gibbs Reflective Cycle. Journal of Engineering Education Transformations, 38(4). https://doi.org/10.16920/jeet/2024/v38i4/25096

Beldar, P. R., Munje, R. K., & Kadbhane, S. V. (2025). Enhancing Engineering Education through Drone Technology Skilling Program: Analyzing the Impact on Program Outcomes. Journal of Engineering Education Transformations, 39(2). https://doi.org/10.16920/jeet/2025/v39i2/25138

Beldar, P., Rakhade, R., Bahiram, M., & Kadbhane, S. (2025). Innovative Coding Teaching Methodologies: A Comprehensive Approach for Diverse Learners. Journal of Engineering Education Transformations, 39(2). https://doi.org/10.16920/jeet/2025/v39i2/25141

Duran, R., Zavgorodniaia, A., & Sorva, J. (2022). Cognitive Load Theory in Computing Education Research: A Review. ACM Transactions on Computing Education, 22(4). https://doi.org/10.1145/3483843

Ellerton, D. P. (2022). On critical thinking and content knowledge: A critique of the assumptions of cognitive load theory. Thinking Skills and Creativity, 43. https://doi.org/10.1016/j.tsc.2021.100975

Evans, P., Vansteenkiste, M., Parker, P., Kingsford-Smith, A., & Zhou, S. (2024). Cognitive Load Theory and Its Relationships with Motivation: A Self-Determination Theory Perspective. In Educational Psychology Review (Vol. 36, Issue 1). https://doi.org/10.1007/s10648-023-09841-2

Fischer, K., Sullivan, A. M., Cohen, A. P., King, R. W., Cockrill, B. A., & Besche, H. C. (2023). Using cognitive load theory to evaluate and improve preparatory materials and study time for the flipped classroom. BMC Medical Education, 23(1). https://doi.org/10.1186/s12909-023-04325-x

Ghanbari, S., Haghani, F., Barekatain, M., & Jamali, A. (2020). A systematized review of cognitive load theory in health sciences education and a perspective from cognitive neuroscience. In Journal of Education and Health Promotion (Vol. 9, Issue 1). https://doi.org/10.4103/jehp.jehp_643_19

Ginns, P., & Leppink, J. (2019). Special Issue on Cognitive Load Theory: Editorial. In Educational Psychology Review (Vol. 31, Issue 2). https://doi.org/10.1007/s10648-019-09474-4

Hanham, J., Castro-Alonso, J. C., & Chen, O. (2023). Integrating cognitive load theory with other theories, within and beyond educational psychology. British Journal of Educational Psychology, 93(S2). https://doi.org/10.1111/bjep.12612

Jong, T. de. (2010). Cognitive load theory, educational research, and instructional design: Some food for thought. Instructional Science, 38(2). https://doi.org/10.1007/s11251-009-9110-0

Kala, N., & Ayas, A. (2023). Effect of instructional design based on cognitive load theory on students' performances and the indicators of element interactivity. Journal of Turkish Science Education, 20(3). https://doi.org/10.36681/tused.2023.027

Kalyuga, S., & Singh, A. M. (2016). Rethinking the Boundaries of Cognitive Load Theory in Complex Learning. Educational Psychology Review, 28(4). https://doi.org/10.1007/s10648-015-9352-0

Leppink, J., & Heuvel, A. van den. (2015). The evolution of cognitive load theory and its application to medical education. In Perspectives on Medical Education (Vol. 4, Issue 3). https://doi.org/10.1007/s40037-015-0192-x

Lopez, S. (2024). Impact of Cognitive Load Theory on the Effectiveness of Microlearning Modules. European Journal of Education and Pedagogy, 5(2). https://doi.org/10.24018/ejedu.2024.5.2.799

Nooijen, C. C. A. van, Koning, B. B. de, Bramer, W. M., Isahakyan, A., Asoodar, M., Kok, E., Merrienboer, J. J. G. van, & Paas, F. (2024). A Cognitive Load Theory Approach to Understanding Expert Scaffolding of Visual Problem-Solving Tasks: A Scoping Review. In Educational Psychology Review (Vol. 36, Issue 1). https://doi.org/10.1007/s10648-024-09848-3

Partarakis, N., & Zabulis, X. (2024). Applying Cognitive Load Theory to eLearning of Crafts. Multimodal Technologies and Interaction, 8(1). https://doi.org/10.3390/mti8010002

Patel, D., & Alismail, A. (2024). Relationship Between Cognitive Load Theory, Intrinsic Motivation and Emotions in Healthcare Professions Education: A Perspective on the Missing Link. Advances in Medical Education and Practice, 15. https://doi.org/10.2147/AMEP.S441405

Sujatha, U., & Rajasekaran, V. (2024). Optimising listening skills: Analysing the effectiveness of a blended model with a top-down approach through cognitive load theory. MethodsX, 12. https://doi.org/10.1016/j.mex.2024.102630

Sweller, J. (2020). Cognitive load theory and educational technology. Educational Technology Research and Development, 68(1). https://doi.org/10.1007/s11423-019-09701-3

Sweller, J. (2022). Cognitive load theory. In International Encyclopedia of Education: Fourth Edition. https://doi.org/10.1016/B978-0-12-818630-5.14020-5

Sweller, J. (2023a). Discussion of the special issue on cognitive load theory. British Journal of Educational Psychology, 93(S2). https://doi.org/10.1111/bjep.12606

Sweller, J. (2023b). The Development of Cognitive Load Theory: Replication Crises and Incorporation of Other Theories Can Lead to Theory Expansion. Educational Psychology Review, 35(4). https://doi.org/10.1007/s10648-023-09817-2

Sweller, J. (2024). Cognitive load theory and individual differences. Learning and Individual Differences, 110. https://doi.org/10.1016/j.lindif.2024.102423

Taylor, T. A. H., Kamel-ElSayed, S., Grogan, J. F., Hussein, I. H., Lerchenfeldt, S., & Mohiyeddini, C. (2022). Teaching in Uncertain Times: Expanding the Scope of Extraneous Cognitive Load in the Cognitive Load Theory. Frontiers in Psychology, 13. https://doi.org/10.3389/fpsyg.2022.665835