Texas A&M University - Commerce

Department of Computer Science

Evaluating the Proficiency of ChatGPT in Undergraduate Data Structures and Algorithms: An Analysis of Standardized Test Performance

Mokshith Ramendra Yaganti

Supervisor: Derek Harter, Ph.D.

A report submitted in partial fulfilment of the requirements of
Texas A&M University - Commerce for the degree of
Master of Science in *Computer Science*

February 5, 2024

Declaration

I, Mokshith ramendra Yaganti, of the Department of Computer Science, Texas A&M University - Commerce, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of TAMUC and public with interest in teaching, learning and research.

Mokshith ramendra Yaganti
February 5, 2024

Abstract

The realm of Artificial Intelligence (AI) is undergoing a significant transformation, primarily driven by the advancements in Large Language Models (LLMs). Among these, ChatGPT has emerged as a standout model, acclaimed for its exceptional ability in conducting multi-turn conversations and showcasing coding proficiency in various programming languages. This study deals with the investigation of the performance of ChatGPT and the impact of prompt engineering, on its effectiveness in solving standardized undergraduate-level data structures and algorithms problems. A novel aspect of this research is the focus on automating the entire evaluation pipeline, including prompt fine-tuning, generating responses from ChatGPT, and systematically testing these responses against a curated set of standard questions. This automation not only streamlines the assessment process but also sets a precedent for analyzing other LLMs in a similar fashion.

The methodology centers on the development and application of tailored prompts designed to maximize ChatGPT's performance in solving complex programming challenges. The study meticulously curates a diverse collection of data structures and algorithms questions, representative of undergraduate coursework. ChatGPT's responses to these prompts are then automatically processed and evaluated against multiple test cases to determine their correctness and efficacy.

Key findings of this research will illuminate the potential of prompt engineering as a crucial factor in enhancing the performance of LLMs in technical domains. The outcomes are expected to provide valuable insights into the capabilities and limitations of ChatGPT in the context of algorithmic problem-solving. Furthermore, the study's automated approach promises scalability and reproducibility, offering a robust framework for future research in LLM performance analysis across various disciplines.

Keywords: ChatGPT, Large Language Models, Prompt Engineering, Data Structures, Algorithms.

Report's total word count: we expect a maximum of 10,000 words (excluding reference and appendices) and about 10 pages. [A good project report can also be written in approximately 5,000 words.]

Acknowledgements

An acknowledgements section is optional. You may like to acknowledge the support and help of your supervisor(s), friends, or any other person(s), department(s), institute(s), etc. If you have been provided specific facility from department/school acknowledged so.

Contents

List of Figures

List of Tables

List of Abbreviations

SMPCS	School of Mathematical, Physical and Computational Sciences

Chapter 1

Introduction

1.1 Background

The proficiency of Large Language Models (LLMs) like ChatGPT in generating working code is intrinsically linked to their understanding and application of data structures and algorithms (DSA). These fundamental components form the backbone of efficient and effective software development. DSA are critical in determining how data is organized, stored, and manipulated within a program, impacting everything from the execution speed to resource utilization. The ability of an AI model to adeptly handle these aspects is indicative of its depth in coding knowledge and its applicability in real-world software development scenarios.

In this context, the capabilities of ChatGPT in DSA are particularly noteworthy. This model has demonstrated a remarkable ability to not only understand and implement standard data structures and algorithms but also to apply them creatively to solve complex problems. The implication of this proficiency is significant; it suggests that LLMs like ChatGPT can be invaluable tools in the software development process, assisting in everything from initial problem analysis to the formulation of efficient algorithmic solutions.

Furthermore, the application of DSA in code generation by LLMs also opens up possibilities for more advanced software development applications. For instance, an AI model proficient in DSA can potentially assist in optimizing existing codebases, refactoring inefficient structures, or even suggesting algorithmic improvements.

In the current landscape of AI-driven code generation, particularly with models like ChatGPT, there is a noticeable challenge in consistently generating code that is both efficient and correct. The generation of correct code is fundamental, as errors in code can lead to significant issues in software development, ranging from minor bugs to critical system failures. While ChatGPT exhibits a strong capacity for code generation especially in DSA, as evidenced in the work by Arefin et al. (2023), its performance can vary, especially in terms of efficiency and adherence to best practices in DSA. This variability underscores the necessity for more refined techniques in interacting with these models to elicit the highest quality of code output.

Prompt engineering emerges as a crucial element in this context. It plays a critical role in harnessing their full potential, especially in code generation. It involves meticulously crafting input prompts to effectively communicate the requirements and constraints of a given problem to the AI model. This practice is particularly essential in programming scenarios where the quality of output is directly influenced by how the problem is presented to the model, i.e., the clarity

and specificity of instructions can significantly influence the accuracy and utility of the generated code. A well-engineered prompt can lead to solutions that are not only syntactically and logically correct but also optimized in terms of performance and resource management. The significance of prompt design in obtaining optimal results from generative tasks has been highlighted in the research by Chen et al. (2021) on GPT models.

The implications of a system proficient in accurately generating working, logical code are far-reaching. In the future, such capabilities could revolutionize software development, enabling faster deployment of robust and sophisticated applications. Furthermore, as noted by Chen et al. (2021) in their analysis of code generation models, these advancements could democratize programming, allowing individuals with limited coding expertise to develop software solutions through intuitive, natural language instructions. This could potentially lead to a surge in innovation, as barriers to software creation are lowered, and a broader range of perspectives are brought into the technology development process.

The current research aims to delve deeper into ChatGPT's capabilities in DSA. It aims to assess and quantify the impact of prompt engineering on ChatGPT's performance in generating algorithmic solutions. By evaluating the model's performance in this domain, the study seeks to shed light on the extent to which ChatGPT can accurately and effectively generate code solutions that are not just correct in their logic, but also optimal in their use of data structures and algorithms. This exploration is pivotal, as it directly relates to the practical utility of LLMs in software engineering, a field where efficiency and optimization are paramount.

1.2 Problem statement

The central problem this study addresses is the assessment of the impact of prompt engineering on the performance of ChatGPT, specifically in solving undergraduate-level data structures and algorithms (DSA) questions. The effectiveness of Large Language Models (LLMs) like ChatGPT in code generation has been increasingly recognized. However, their ability to consistently produce efficient and correct solutions in the context of complex DSA problems remains an area requiring deeper exploration. This research hypothesizes that through strategic prompt engineering, the proficiency of ChatGPT in generating accurate and optimized solutions to DSA challenges can be significantly enhanced.

Prompt engineering, in this context, refers to the deliberate design and structuring of input prompts to guide ChatGPT in understanding and effectively responding to the intricacies of DSA problems. The hypothesis is grounded in the premise that the manner in which a problem is presented to an LLM can profoundly influence the quality of the generated solution. This hypothesis aligns with the findings of Radford et al. (2019), who highlighted the importance of prompt design in achieving desired outcomes from generative AI models. The study aims to empirically test this hypothesis by systematically varying the prompts used to interact with ChatGPT and evaluating the resulting code's correctness, efficiency, and adherence to DSA best practices.

The significance of this investigation lies not only in its potential to enhance the understanding of prompt engineering as a tool for optimizing LLM performance but also in its broader implications for the field of AI-driven software development. By establishing a clear correlation between prompt engineering and the quality of ChatGPT-generated code, the study seeks to contribute to the development of more effective methodologies for leveraging LLMs in technical and educational

settings.

1.3 Aims and objectives

The overarching aim of this project is to conduct a comprehensive evaluation of ChatGPT's capabilities in solving data structures and algorithms problems typically encountered at the undergraduate level. This aim encompasses several specific objectives:

Development of Tailored Prompts: One of the primary objectives is to develop tailored prompts that enhance ChatGPT's problem-solving skills in data structures and algorithms. This involves designing prompts that not only accurately describe the problem but also guide the model towards optimal problem-solving approaches. The effectiveness of these prompts will be gauged by their ability to elicit accurate, efficient, and sophisticated solutions from ChatGPT.

Curating a Comprehensive Standard Test Set: The project will involve curating a comprehensive set of standard undergraduate-level questions in data structures and algorithms. The selected test set for this research includes the Leetcode 75, Blind 75 Leetcode questions, and top interview 150 Leetcode questions. These questions are chosen for their relevance, diversity, and the depth they provide in covering key concepts within the field.

Automation of the Evaluation Process: Automating the evaluation of ChatGPT's responses is another critical objective. This will be achieved by integrating ChatGPT with the LeetCode API, allowing for an efficient and systematic assessment of the solutions' correctness and practical applicability. Automation will ensure consistency in evaluation and enable the handling of a large volume of test cases.

Comparison of Effectiveness: The project aims to compare the effectiveness of ChatGPT's responses with and without prompt engineering. This comparative analysis will help ascertain the impact of prompt engineering on the model's performance. It will involve analyzing the solutions generated by ChatGPT under different prompting conditions to evaluate how variations in prompt design influence the accuracy and quality of the generated code.

Through these objectives, the project seeks to provide a detailed analysis of how effectively ChatGPT can be utilized in the domain of data structures and algorithms, and the extent to which prompt engineering can enhance its performance. The results are expected to offer valuable insights into the practical applications of LLMs in technical education and software development.

1.4 Solution approach

The methodology of this project is structured around a blend of prompt engineering and automated evaluation techniques, designed to rigorously test ChatGPT's proficiency in solving data structures and algorithms problems.

By employing these methodical and automated approaches outlined below, the study aims to provide a detailed analysis of ChatGPT's problem-solving abilities in the context of data structures and algorithms. The findings are expected to offer significant insights into the potential of LLMs in technical education and software development, particularly in the realm of algorithmic thinking and coding proficiency.

1.4.1 Automated System Development and Integration

A key component of our methodology involves developing an automated system that interfaces with both the ChatGPT and LeetCode APIs. This system will automatically fetch questions from LeetCode (Leetcode 75, Blind 75 Leetcode questions, and top interview 150 Leetcode questions), using a combination of the LeetCode Rust API and web scraping methods. These questions, representative of common undergraduate-level data structures and algorithms challenges, form the basis of our evaluation set.

1.4.2 Solution Generation and Submission

Once a question is fetched, it is passed to the ChatGPT API to generate a solution. This process is repeated multiple times for each question, with each iteration using a different set of prompts. These prompts are strategically designed to guide ChatGPT towards generating more effective solutions. Each generated solution is then submitted back to LeetCode through its API, enabling us to assess its correctness and efficiency.

1.4.3 Evaluation Metrics

The primary metrics for evaluating each solution are the number of test cases passed and the run time. The number of test cases passed is a direct indicator of the solution's correctness, reflecting how well the code meets the problem's requirements. The run time, on the other hand, provides insight into the efficiency and optimization of the solution. Together, these metrics are critical for evaluating the effectiveness of different prompts and the overall capability of ChatGPT in solving complex DSA questions.

1.4.4 Comprehensive Testing Across Standard Test Questions

This entire process, from question retrieval to solution evaluation, is systematically repeated for all the standard test questions outlined in the Aims & Objectives section. This comprehensive approach ensures a thorough assessment of ChatGPT's capabilities and the impact of various prompt engineering strategies on its performance.

1.5 Summary of contributions and achievements

Describe clearly what you have done/created/achieved and what the major results and their implications are.

1.6 Organization of the report

Describe the outline of the rest of the report here. Let the reader know what to expect ahead in the report. Describe how you have organized your report.

Example: how to refer a chapter, section, subsection. This report is organised into seven chapters. Chapter 2 details the literature review of this project. In Section 3...

Note: Take care of the word like "Chapter," "Section," "Figure" etc. before the LaTeXcommand \ref{}. Otherwise, a sentence will be confusing. For example, In 2 literature review is described.

In this sentence, the word "Chapter" is missing. Therefore, a reader would not know whether 2 is for a Chapter or a Section or a Figure.

Chapter 2

Literature Review

Our investigation delves into two primary areas of research concerning ChatGPT: its coding capabilities and its performance in non-coding tasks. This review delineates these areas and positions our study within the broader research landscape.

Previous studies have extensively explored ChatGPT's coding skills. Notably, Noever and McKee ([Noever et al.](https://arxiv.org/abs/2301.13382)) evaluated ChatGPT's proficiency in performing CRUD operations using datasets like Iris, Titanic, Boston Housing, and Faker. Their findings indicated that ChatGPT could effectively emulate a Python interpreter, generating code and outputs autonomously. This demonstrates ChatGPT's capability to handle structured data and execute CRUD operations efficiently. Biswas et al. ([Biswas et al.](https://doi.org/10.58496/MJCSC/2023/002)) further explored ChatGPT's role as a programming aide, highlighting its assistance in code completion, debugging, and refactoring tasks.

A study paralleling our methodology ([Bubeck et al.](https://arxiv.org/abs/2303.12712)) tasked ChatGPT with creating Python functions, initially using the HumanEval dataset and later a set of LeetCode problems. This research benchmarked GPT-4 against other models and human performance, revealing GPT-4's superior capabilities. Another comprehensive analysis ([Tian et al.](https://arxiv.org/abs/2304.11938)) examined ChatGPT's code translation, correction, and comprehension skills, offering a nuanced view of its coding proficiency but also noting occasional inaccuracies and inefficiencies.

Moreover, a study by Feng et al. ([Feng et al.](https://arxiv.org/abs/2304.11938)) employed a crowdsourcing approach to evaluate ChatGPT's code generation capabilities, analyzing social media discourse to glean insights into programming language preferences, usage scenarios, and common errors in generated code snippets.

Our research aims to extend these findings by focusing on a broader set of coding challenges and evaluating ChatGPT's coding quality across various programming tasks, particularly in data structures and algorithms. This approach allows us to contribute novel insights into ChatGPT's applicability and efficiency in software development.

The application of ChatGPT extends beyond coding. Gilson et al. ([Gilson et al.](https://doi.org/10.2196/45312)) assessed ChatGPT's ability to answer medical licensing exam questions, noting its proficiency in recall questions but limitations in reasoning. Similarly, studies on ChatGPT's mathematical skills ([Frieder et al.](https://arxiv.org/abs/2301.13867); [Pardos and Bhandari](https://arxiv.org/abs/2302.06871)) revealed its creative reasoning capabilities but highlighted deficiencies in critical reasoning and technical accuracy.

Additionally, research by Zhuo et al. ([Zhuo et al.](https://arxiv.org/abs/2301.12867)) and Bang et al. ([Bang et al.](https://arxiv.org/abs/2302.04023)) investigated ChatGPT's linguistic abilities and ethical implications, uncovering challenges related to bias, reliability, and understanding of non-Latin scripts.

While these studies offer valuable perspectives on ChatGPT's utility in diverse domains, our research is distinctly focused on optimizing ChatGPT's performance in solving data structures and algorithms problems through strategic prompt engineering. By focusing on this niche, we aim to enhance the understanding of how strategic prompt design can optimize ChatGPT's performance in technical education and software development. By examining the impact of prompt engineering on ChatGPT's performance, our research seeks to bridge gaps in the current literature and contributes to the field by exploring methods to harness ChatGPT's potential more effectively, particularly in areas requiring sophisticated algorithmic thinking and problem-solving skills.

2.1 Critique of the review

Describe your main findings and evaluation of the literature.

2.2 Summary

Write a summary of this chapter

Chapter 3

Methodology

We mentioned in Chapter 1 that a project report's structure could follow a particular paradigm. Hence, the organization of a report (effectively the Table of Content of a report) can vary depending on the type of project you are doing. Check which of the given examples suit your project. Alternatively, follow your supervisor's advice.

3.1 Examples of the sections of a methodology chapter

A general report structure is summarised (suggested) in Table 3.1. Table 3.1 describes that, in general, a typical report structure has three main parts: (1) front matter, (2) main text, and (3) end matter. The structure of the front matter and end matter will remain the same for all the undergraduate final year project report. However, the main text varies as per the project's needs.

3.1.1 Example of a software/Web development main text structure

Notice that the "methodology" Chapter of Software/Web development in Table 3.2 takes a standard software engineering paradigm (approach). Alternatively, these suggested sections can be the chapters of their own. Also, notice that "Chapter 5" in Table 3.2 is "Testing and Validation" which is different from the general report template mentioned in Table 3.1. Check with your supervisor if in doubt.

3.1.2 Example of an algorithm analysis main text structure

Some project might involve the implementation of a state-of-the-art algorithm and its performance analysis and comparison with other algorithms. In that case, the suggestion in Table 3.3 may suit you the best.

3.1.3 Example of an application type main text structure

If you are applying some algorithms/tools/technologies on some problems/datasets/etc., you may use the methodology section prescribed in Table 3.4.

Table 3.1: Undergraduate report template structure

Frontmatter		Title Page Abstract Acknowledgements Table of Contents List of Figures List of Tables List of Abbreviations
Main text	Chapter 1	Introduction
	Chapter 2	Literature Review
	Chapter 3	Methodology
	Chapter 4	Results
	Chapter 5	Discussion and Analysis
	Chapter 6	Conclusions and Future Work
	Chapter 7	Refection
End matter		References Appendices (Optional) Index (Optional)

Table 3.2: Example of a software engineering-type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Requirements specifications
		Analysis
		Design
		Implementations
Chapter 4	Testing and Validation	
Chapter 5	Results and Discussion	
Chapter 6	Conclusions and Future Work	
Chapter 7	Reflection	

3.1.4 Example of a science lab-type main text structure

If you are doing a science lab experiment type of project, you may use the methodology section suggested in Table 3.5. In this kind of project, you may refer to the "Methodology" section as "Materials and Methods."

Table 3.3: Example of an algorithm analysis type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Algorithms descriptions
		Implementations
		Experiments design
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

Table 3.4: Example of an application type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Methodology	
		Problems (tasks) descriptions
		Algorithms/tools/technologies/etc. descriptions
		Implementations
		Experiments design and setup
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

Table 3.5: Example of a science lab experiment-type report structure

Chapter 1	Introduction	
Chapter 2	Literature Review	
Chapter 3	Materials and Methods	
		Problems (tasks) description
		Materials
		Procedures
		Implementations
		Experiment set-up
Chapter 4	Results	
Chapter 5	Discussion and Analysis	
Chapter 6	Conclusion and Future Work	
Chapter 7	Reflection	

3.2 Example of an Equation in LaTeX

Eq. 3.1 [note that this is an example of an equation's in-text citation] is an example of an equation in LaTeX. In Eq. (3.1), s is the mean of elements $x_i \in \mathbf{x}$:

$$s = \frac{1}{N} \sum_{i=1}^{N} x_i. \tag{3.1}$$

Have you noticed that all the variables of the equation are defined using the **in-text** maths command $.$, and Eq. (3.1) is treated as a part of the sentence with proper punctuation? Always treat an equation or expression as a part of the sentence.

3.3 Example of a Figure in LaTeX

Figure 3.1 is an example of a figure in LaTeX. For more details, check the link:
wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions.
Keep your artwork (graphics, figures, illustrations) clean and readable. At least 300dpi is a good resolution of a PNG format artwork. However, an SVG format artwork saved as a PDF will produce the best quality graphics. There are numerous tools out there that can produce vector graphics and let you save that as an SVG file and/or as a PDF file. One example of such a tool is the "Flow algorithm software". Here is the link for that: flowgorithm.org.

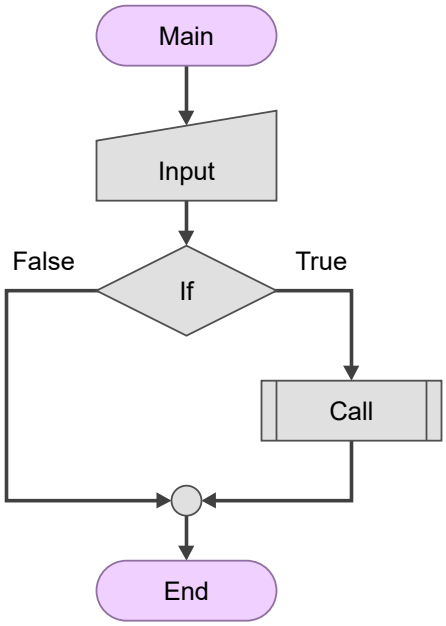

Figure 3.1: Example figure in LaTeX.

3.4 Example of an algorithm in LaTeX

Algorithm 1 is a good example of an algorithm in LaTeX.

Algorithm 1 Example caption: sum of all even numbers

Input: $\mathbf{x} = x_1, x_2, \ldots, x_N$
Output: $EvenSum$ (Sum of even numbers in $\mathbf{x}$)

1: **function** EvenSummation($\mathbf{x}$)
2: $EvenSum \leftarrow 0$
3: $N \leftarrow length(\mathbf{x})$
4: **for** $i \leftarrow 1$ to N **do**
5: **if** $x_i \mod 2 == 0$ **then** ▷ check if a number is even?
6: $EvenSum \leftarrow EvenSum + x_i$
7: **end if**
8: **end for**
9: **return** $EvenSum$
10: **end function**

3.5 Example of code snippet in LaTeX

Code Listing 3.1 is a good example of including a code snippet in a report. While using code snippets, take care of the following:

- do not paste your entire code (implementation) or everything you have coded. Add code snippets only.
- The algorithm shown in Algorithm 1 is usually preferred over code snippets in a technical/-scientific report.
- Make sure the entire code snippet or algorithm stays on a single page and does not overflow to another page(s).

Here are three examples of code snippets for three different languages (Python, Java, and CPP) illustrated in Listings 3.1, 3.2, and 3.3 respectively.

```
1  import numpy as np
2
3  x  = [0, 1, 2, 3, 4, 5] # assign values to an array
4  evenSum = evenSummation(x) # call a function
5
6  def evenSummation(x):
7      evenSum = 0
8      n = len(x)
9      for i in range(n):
10         if np.mod(x[i],2) == 0: # check if a number is even?
11             evenSum = evenSum + x[i]
12     return evenSum
```

Listing 3.1: Code snippet in LaTeX and this is a Python code example

Here we used the "\clearpage" command and forced-out the second listing example onto the next page.

```java
public class EvenSum{
    public static int evenSummation(int[] x){
        int evenSum = 0;
        int n = x.length;
        for(int i = 0; i < n; i++){
            if(x[i]%2 == 0){ // check if a number is even?
                evenSum = evenSum + x[i];
            }
        }
        return evenSum;
    }
    public static void main(String[] args){
        int[] x  = {0, 1, 2, 3, 4, 5}; // assign values to an array
        int evenSum = evenSummation(x);
        System.out.println(evenSum);
    }
}
```

Listing 3.2: Code snippet in LaTeX and this is a Java code example

```cpp
int evenSummation(int x[]){
    int evenSum = 0;
    int n = sizeof(x);
    for(int i = 0; i < n; i++){
        if(x[i]%2 == 0){ // check if a number is even?
            evenSum = evenSum + x[i];
      }
    }
    return evenSum;
}

int main(){
    int x[]  = {0, 1, 2, 3, 4, 5}; // assign values to an array
    int evenSum = evenSummation(x);
    cout<<evenSum;
    return 0;
}
```

Listing 3.3: Code snippet in LaTeX and this is a C/C++ code example

3.6 Example of in-text citation style

3.6.1 Example of the equations and illustrations placement and reference in the text

Make sure whenever you refer to the equations, tables, figures, algorithms, and listings for the first time, they also appear (placed) somewhere on the same page or in the following page(s). Always make sure to refer to the equations, tables and figures used in the report. Do not leave them without an **in-text citation**. You can refer to equations, tables and figures more them once.

3.6.2 Example of the equations and illustrations style

Write **Eq.** with an uppercase "Eq" for an equation before using an equation number with (\eqref{.}). Use "Table" to refer to a table, "Figure" to refer to a figure, "Algorithm" to

refer to an algorithm and "Listing" to refer to listings (code snippets). Note that, we do not use the articles "a," "an," and "the" before the words Eq., Figure, Table, and Listing, but you may use an article for referring the words figure, table, etc. in general.

For example, the sentence "A report structure is shown in **the** Table 3.1" should be written as "A report structure is shown **in** Table 3.1."

3.7 Summary

Write a summary of this chapter.

Note: In the case of **software engineering** project a Chapter "**Testing and Validation**" should precede the "Results" chapter. See Section 3.1.1 for report organization of such project.

Chapter 4

Results

The results chapter tells a reader about your findings based on the methodology you have used to solve the investigated problem. For example:

- If your project aims to develop a software/web application, the results may be the developed software/system/performance of the system, etc., obtained using a relevant methodological approach in software engineering.
- If your project aims to implement an algorithm for its analysis, the results may be the performance of the algorithm obtained using a relevant experiment design.
- If your project aims to solve some problems/research questions over a collected dataset, the results may be the findings obtained using the applied tools/algorithms/etc.

Arrange your results and findings in a logical sequence.

4.1 A section

...

4.2 Example of a Table in LaTeX

Table 4.1 is an example of a table created using the package LaTeX"booktabs." do check the link: wikibooks.org/wiki/LaTeX/Tables for more details. A table should be clean and readable. Unnecessary horizontal lines and vertical lines in tables make them unreadable and messy. The example in Table 4.1 uses a minimum number of liens (only necessary ones). Make sure that the top rule and bottom rule (top and bottom horizontal lines) of a table are present.

Table 4.1: Example of a table in LaTeX

Bike		
Type	Color	Price (£)
Electric	black	700
Hybrid	blue	500
Road	blue	300
Mountain	red	300
Folding	black	500

4.3 Example of captions style

- The **caption of a Figure (artwork) goes below** the artwork (Figure/Graphics/illustration). See example artwork in Figure 3.1.
- The **caption of a Table goes above** the table. See the example in Table 4.1.
- The **caption of an Algorithm goes above** the algorithm. See the example in Algorithm 1.
- The **caption of a Listing goes below** the Listing (Code snippet). See example listing in Listing 3.1.

4.4 Summary

Write a summary of this chapter.

Chapter 5

Discussion and Analysis

Depending on the type of project you are doing, this chapter can be merged with "Results" Chapter as " Results and Discussion" as suggested by your supervisor.

In the case of software development and the standalone applications, describe the significance of the obtained results/performance of the system.

5.1 A section

Discussion and analysis chapter evaluates and analyses the results. It interprets the obtained results.

5.2 Significance of the findings

In this chapter, you should also try to discuss the significance of the results and key findings, in order to enhance the reader's understanding of the investigated problem

5.3 Limitations

Discuss the key limitations and potential implications or improvements of the findings.

5.4 Summary

Write a summary of this chapter.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Typically a conclusions chapter first summarizes the investigated problem and its aims and objectives. It summaries the critical/significant/major findings/results about the aims and objectives that have been obtained by applying the key methods/implementations/experiment set-ups. A conclusions chapter draws a picture/outline of your project's central and the most signification contributions and achievements.

A good conclusions summary could be approximately 300–500 words long, but this is just a recommendation.

A conclusions chapter followed by an abstract is the last things you write in your project report.

6.2 Future work

This section should refer to Chapter 4 where the author has reflected their criticality about their own solution. The future work is then sensibly proposed in this section.

Guidance on writing future work: While working on a project, you gain experience and learn the potential of your project and its future works. Discuss the future work of the project in technical terms. This has to be based on what has not been yet achieved in comparison to what you had initially planned and what you have learned from the project. Describe to a reader what future work(s) can be started from the things you have completed. This includes identifying what has not been achieved and what could be achieved.

A good future work summary could be approximately 300–500 words long, but this is just a recommendation.

Chapter 7

Reflection

Write a short paragraph on the substantial learning experience. This can include your decision-making approach in problem-solving.

Some hints: You obviously learned how to use different programming languages, write reports in LaTeXand use other technical tools. In this section, we are more interested in what you thought about the experience. Take some time to think and reflect on your individual project as an experience, rather than just a list of technical skills and knowledge. You may describe things you have learned from the research approach and strategy, the process of identifying and solving a problem, the process research inquiry, and the understanding of the impact of the project on your learning experience and future work.

Also think in terms of:

- what knowledge and skills you have developed
- what challenges you faced, but was not able to overcome
- what you could do this project differently if the same or similar problem would come
- rationalize the divisions from your initial planed aims and objectives.

A good reflective summary could be approximately 300–500 words long, but this is just a recommendation.

Note: The next chapter is "**References**," which will be automatically generated if you are using BibTeX referencing method. This template uses BibTeX referencing. Also, note that there is difference between "References" and "Bibliography." The list of "References" strictly only contain the list of articles, paper, and content you have cited (i.e., refereed) in the report. Whereas Bibliography is a list that contains the list of articles, paper, and content you have cited in the report plus the list of articles, paper, and content you have read in order to gain knowledge from. We recommend to use only the list of "References."

References

Arefin, S. E., Heya, T. A., Al-Qudah, H., Ineza, Y. and Serwadda, A. (2023), 'Unmasking the giant: A comprehensive evaluation of chatgpt's proficiency in coding algorithms and data structures'.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G. et al. (2021), 'Evaluating large language models trained on code', *arXiv preprint arXiv:2107.03374* .

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. et al. (2019), 'Language models are unsupervised multitask learners', *OpenAI blog* **1**(8), 9.

Appendix A

An Appendix Chapter (Optional)

Some lengthy tables, codes, raw data, length proofs, etc. which are **very important but not essential part** of the project report goes into an Appendix. An appendix is something a reader would consult if he/she needs extra information and a more comprehensive understating of the report. Also, note that you should use one appendix for one idea.

An appendix is optional. If you feel you do not need to include an appendix in your report, avoid including it. Sometime including irrelevant and unnecessary materials in the Appendices may unreasonably increase the total number of pages in your report and distract the reader.

Appendix B

An Appendix Chapter (Optional)

...