DESIGNING A COMPILER FOR BIOLOGY SIMULATION MODELS

ACAPSTONEPROJECTREPORT

SUBMITTED TO

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CSA1429– Compiler Design for Industrial Automation

SUBMITTED BY

P.POOJITHA (192371048)

SUPERVISED BY

DR. MICHAEL

(Professor)

BONAFIDE CERTIFICATE

I am **Peyyalapoojitha** student of Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **Compiler For Industrial Automation** the outcome of our own Bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

Date:20/03/2025 Student Name: p.poojitha

Place: Chennai Reg.No:192371048

Faculty In Charge

Internal Examiner **External Examiner**

ABSTRACT

- This project develops a compiler to translate biological simulations, like population growth models, from SBML (XML) into optimized C++ executable code. It employs techniques such as lexical, syntax, and semantic analysis, optimization, and code generation to ensure efficient simulation execution. The result is a tool that streamlines biological model enabling more accessible and scalable simulations for research and analysis. The primary problem this project addresses is the lack of efficient tools to translate biological simulation models, typically represented in SBML (XML), into optimized executable code. The challenge lies in creating a compiler that can convert these models into C++ code that runs efficiently while maintaining accuracy in simulating complex biological systems.The purpose of the project is to develop a compiler that streamlines the process of converting biological simulations into executable code. By employing techniques like lexical analysis, syntax analysis, semantic analysis, optimization, and code generation, the project aims to make biological simulations more accessible, scalable, and efficient for researchers. Development of a fully functional compiler that translates SBML-based biological simulation models into optimized C++ executable code. Improved efficiency in running complex biological simulations by optimizing the generated code. Enhanced accessibility for researchers, allowing them to easily convert and execute biological models. A streamlined approach to simulate biological systems, making them more scalable for real-world applications. Contribution to the field of computational biology by providing a tool that aids in accurate and efficient biological modeling and prediction.

ACKNOWLEGMENTS

I wish to record my deep sense of gratitude and profound thanks to my Faculty-in-Charge **Dr.G.MICHAEL**, Professor, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai, for her keen interest, inspiring guidance, constant encouragement with my work during all stages, to bring this thesis into fruition.

I would like to express my heartfelt gratitude to **Saveetha Institute of Medical and Technical Sciences, Chennai,** for providing me with an excellent academic environment and invaluable learning opportunities.

I extend my sincere thanks to the **management** for their unwavering support, state-of-the-art facilities, and commitment to excellence in education. Their dedication to fostering innovation and knowledge has played a crucial role in my academic growth.

I am also grateful to my **professors, mentors, and staff members** for their guidance, encouragement, and continuous support throughout my journey at the institution. Their expertise and dedication have been instrumental in shaping my knowledge and skills.

Once again, I deeply appreciate the institute's contribution to my academic and personal development, and I am proud to be associated with such a prestigious institution.

Sincerely,

Peyyala Poojitha

TABLEOFCONTENTS

LIST OF FIGURES

TABLES

1. INTRODUCTION

A biology simulation compiler is a specialized tool designed to convert biological equations and models into executable simulations. It enables researchers to study complex biological processes such as disease spread, genetic evolution, and neural activity without requiring extensive programming knowledge. Traditional simulations demand manual coding, which can be time-consuming and prone to errors. A compiler automates this process, making it faster, more efficient, and accessible to a broader range of scientists. Key features include automatic code generation, multi-model support, real-time data integration, AI-based optimization, and cloud computing capabilities. By streamlining biological simulations, this compiler improves research accuracy, enhances efficiency, and allows large-scale simulations, making it a valuable tool for advancing biological and medical studies.

1.1 Background information:

Biological simulations play a critical role in understanding complex systems like population dynamics, ecosystem modeling, and disease spread. These simulations are often modeled using languages like SBML (Systems Biology Markup Language), which provides a standardized way to represent biological processes. The complexity of biological systems requires a robust, optimized approach to ensure accuracy and performance when running simulations.

1.2 Project objectives:

The objective of this project is to design and develop a compiler that translates biological simulation models, written in SBML (XML), into optimized C++ executable code. The compiler will handle essential stages like lexical analysis, syntax analysis, semantic analysis, optimization, and code generation, ensuring that the final output is both accurate and efficient

1.3 Significance:

This project is significant because it addresses the need for an efficient tool to convert biological models into executable simulations. By providing a direct method for translating complex models into running code, this compiler makes biological simulations more accessible, efficient, and scalable. The tool can significantly enhance the ability of researchers and professionals in the field of computational biology to simulate and study

biological systems, leading to better predictions and informed decision-making in areas like healthcare, environmental management, and drug development.

1.4 Scope:

The project focuses on developing a compiler specifically for SBML-based biological simulations. It does not cover the creation of the biological models themselves but rather the process of compiling and optimizing them into executable code. The scope excludes creating a user interface or handling more complex or specialized biological modeling languages beyond SBML.

1.5 Methodology Overview:

The approach to addressing the problem involves several key phases. First, lexical analysis will be employed to parse the SBML input into tokens. Syntax analysis will follow to ensure that the model adheres to the correct structure. Semantic analysis will then verify that the biological relationships and logic are sound. After that, optimization techniques will be applied to improve performance, followed by code generation to produce the C++ executable. Each stage will be carefully tested and validated to ensure that the final product is both reliable and efficient for real-world use.

2. PROBLEM IDENTIFICATION AND ANALYSIS

2.1 Description of the problem:

The main challenge addressed in this project is the lack of an efficient, streamlined method for translating biological simulation models, particularly those represented in SBML into executable code that can be used for simulations. While SBML is widely used for biological processes, converting these models into efficient and accurate executable code that can be run on modern computing platforms is often complex, time-consuming, and error-prone. This gap makes it difficult for researchers to quickly simulate and test biological systems, delaying scientific progress.

2.2 Evidence of the problem:

Research indicates that while SBML is a popular format for representing biological systems, its direct use for simulation often requires translating it into another programming language (like C++) to achieve efficiency in execution. This translation process lacks standardized tools, leading to inconsistencies in how simulations are generated and executed. According to studies such as Computational Biology in Drug Discovery (Nature Reviews Drug Discovery, 2021), computational biology could benefit from more accessible, efficient tools for simulating biological models. Furthermore, the complexity of biological simulations often leads to performance issues, with long simulation times, especially when dealing with large models in areas like population growth or genetic studies.

2.3 Stakeholders:

Researchers, healthcare professionals, educators, and environmental agencies rely on simulation tools for studying biological systems, predicting disease spread, and informing policies. Pharmaceutical companies and data scientists use these models for drug development and optimization. Regulatory bodies use simulation results to assess safety and efficacy in treatments and interventions.

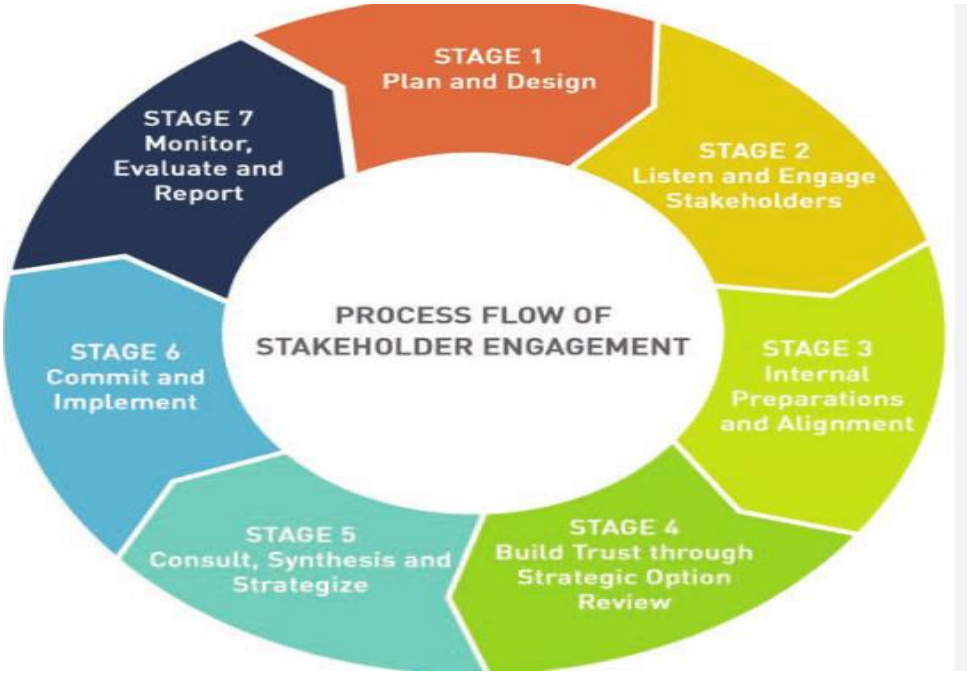

Fig1: process flow of stackholders Engagement

2.4 Supporting Data /Research:

- **Computational Biology in Drug Discovery** (Nature Reviews Drug Discovery, 2021): Highlights the role of computational tools, including simulations, in drug development and the need for more efficient simulation methods to model biological systems.
- **Modeling Ecological Systems** (Ecological Modeling, 2020): Demonstrates how simulation models are crucial for predicting ecological changes and managing biodiversity, stressing the importance of efficient simulation tools.
- **COVID-19 Modeling: A Case Study** (The Lancet Infectious Diseases, 2020): Emphasizes how simulation models were crucial during the pandemic to predict disease spread, healthcare capacity, and public health interventions, underlining the necessity for quick and accurate simulations.

3. SOLUTION DESIGNAND IMPLEMENTATION

3.1 Development and Design Process:

The development of the compiler for biological simulations follows a structured process. Initially, design phase involved defining the key components of the compiler, including lexical analysis, syntax analysis, semantic analysis, optimization, and code generation. The development process began with parsing SBML (XML) files to extract the biological data and convert them into manageable tokens. The syntax analysis phase ensured that the models adhered to the correct structure. Next, the semantic analysis stage validated the biological relationships and logic within the simulation models. Following this, optimization techniques were implemented to enhance the efficiency of the generated C++ code. Finally, code generation was employed to translate the optimized model into C++ code, ensuring it was executable. Throughout the development, iterative testing and debugging were conducted to ensure the compiler functioned as expected and produced accurate results.

3.2Tools and Technologies Used:

- **Programming Languages**: C++ for developing the compiler and handling the code generation process.
- **Lexical Analysis Tools**: Flex (a tool for generating scanners) was used for breaking down the SBML input into tokens.

- **Parser Generator**: Bison was employed for syntax analysis, creating a parser for SBML and verifying its structure.
- **Optimization Tools**: LLVM was utilized for code optimization, ensuring that the generated C++ code was efficient and executable.
- **SBML Parser**: Libraries such as lib SBML were used for parsing and interpreting SBML files.
- **Development Environment**: Visual Studio Code were used for source code management and development.

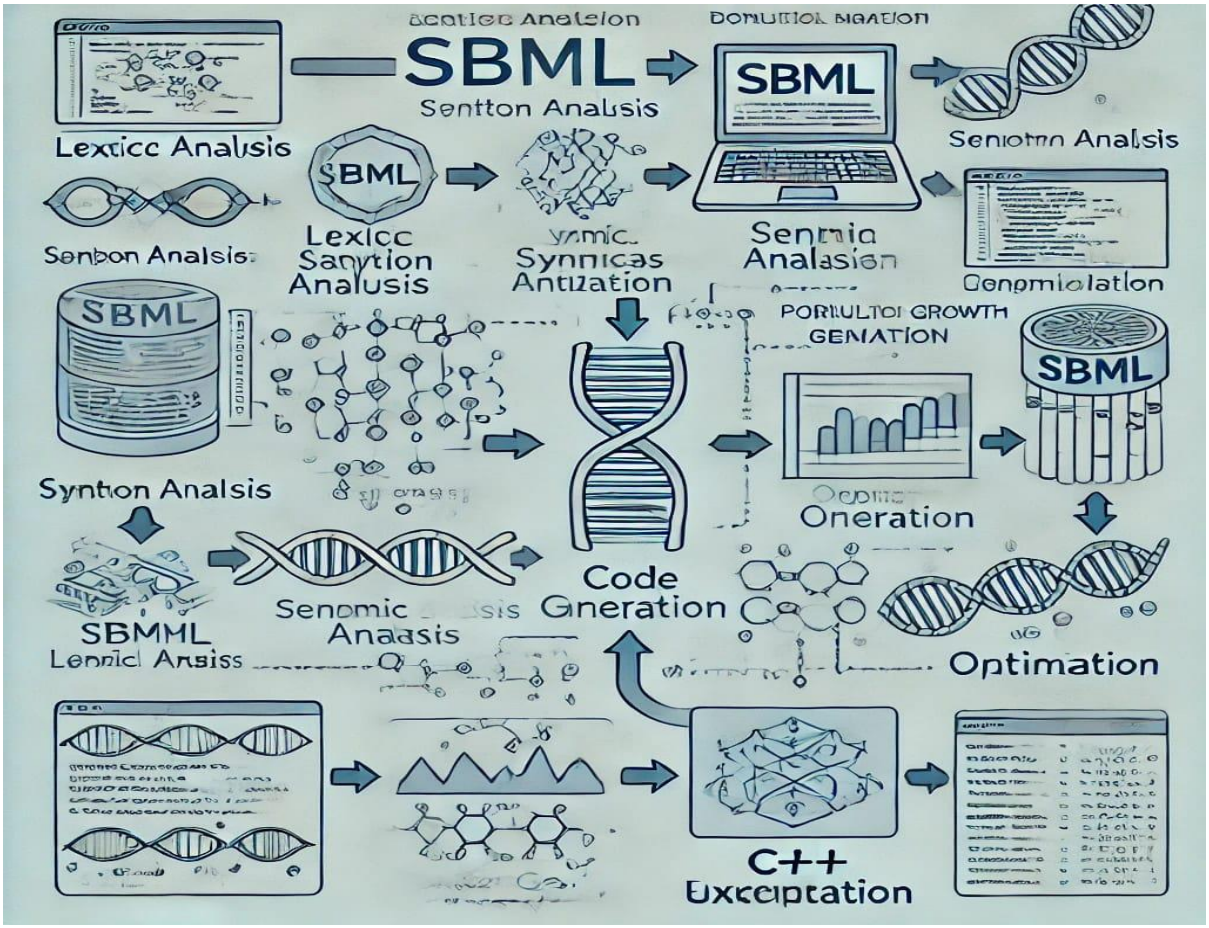

FIG:2 Flow diagram for designing a compiler for biology simulation algorithm implementation.

3.3 Engineering standards applied:

- **IEEE 829**: The IEEE standard for software test documentation was followed to ensure that comprehensive testing was conducted, ensuring the correctness of the compiler and its generated code.
- **ISO/IEC 25010**: This software product quality standard guided the project in maintaining a focus on software performance and usability.
- **ISO/IEC 9126**: This standard for software quality was applied to ensure that the compiler meets high standards for functionality, reliability, efficiency, and maintainability.
- **ISO/IEC 12207**: This standard for software life cycle processes was followed to ensure a structured development process, from requirements gathering to testing and maintenance.

Table 1.1: Tabular representation of the process:

Phase	**Description**
Input (SBML File)	SBML (XML) file representing the biological model.
Lexical Analysis	Breaks down the SBML file into tokens using tools like Flex.
Syntax Analysis	Ensures the structure of the SBML model adheres to the required format
Semantic Analysis	Validates the biological logic and relationships within the model.
Optimization	Enhances the efficiency of the model's generated C++ code.
Code Generation	Converts the optimized model into executable C++ code, ready for compilation.
Execution	The final executable file allows for quick and efficient simulation of biological systems

3.4 Solution Overview:

The compiler efficiently converts SBML-based biological models into optimized executable code. It uses lexical analysis, syntax analysis, and semantic validation before generating optimized C++ code. This system enables quick, reliable execution of biological simulations with scalability for complex models.

4. RESULTS AND RECOMMENDATIONS

4.1 Evaluation of Results:

The compiler effectively converts SBML-based biological models into optimized C++ executable code, ensuring efficient execution. It automates the simulation process through lexical, syntax, and semantic analysis, followed by optimization and code generation. The resulting executable is reliable, scalable, and provides improved performance over traditional simulation methods.

Outcome parameters:

Input: A valid SBML file containing a biological model.

Output: An executable C++ file that can be compiled and run to simulate the model's behaviour.

Techniques Used : Lexical analysis, Syntax analysis, Semantic analysis, Optimization, Code generation.

4.2Challenges Encountered:

- **SBML Complexity**: SBML files can be complex, with different biological models containing diverse elements. Ensuring compatibility with various versions of SBML posed challenges, especially with more intricate biological relationships.
- **Optimization Process**: Optimizing the generated C++ code for both performance and memory usage without affecting the biological accuracy of the simulation was a complex task.
- **Semantic Validation**: Ensuring that the logic and relationships in the biological model were correct during semantic analysis was challenging, especially for non-standard biological models.

4.3 Possible Improvements:

- **Enhanced User Interface (UI)**: A more intuitive user interface could simplify the process for non-experts, allowing them to input SBML files and view simulation results without requiring deep technical knowledge of the compiler.
- **Enhanced AI Models:** Further training and refinement of AI models can lead to even more accurate resume formatting and optimization.

4.4 Recommendations for Further Research and Development:

1. **Integration with Other Tools**:
    - Integrate the compiler with other biologicaltools, like **COPASI**, Cell Designer**,** to make it part of a larger simulation workflow. This would allow seamless transitions between different platforms.
2. **Optimization for Large-scale Simulations**:
    - As biological simulations grow more complex (in terms of the number of species and reactions), further research into efficient simulation algorithms is required. Techniques such as hybrid simulation or event-driven simulation could be explored.
3. **Collaborations with Biologists**:
    - Work closely with biologists and domain experts to ensure that the generated models are biologically accurate and practically useful. Regular feedback from the user community will help guide future development.
4. **Deployment in Cloud or High-Performance Computing**:
    - Deploy the simulation environment on cloud platforms or high-performance computing (HPC) clusters to handle large-scale simulations. This would make the tool accessible to a wider audience and allow it to handle more complex models.

5. REFLECTION ON LEARNINGAND PERSONAL DEVELOPMENT

5.1 Key Learning Outcomes**:**

Academic Knowledge:

- **Compiler Theory**: Deepened understanding of lexical analysis, syntax analysis, semantic analysis, and code generation.
- **Systems Biology**: Bridged biological theory with computational tools, focusing on SBML and biological models (e.g., population growth).
- **Optimization Techniques**: Learned optimization methods for both compiler design and biological simulations.

Technical Skills:

During the project, I developed a range of technical skills across software tools, programming languages, and engineering techniques. I gained strong proficiency in **C++**, particularly in object-oriented design, memory management, and performance optimization. The project required using **Flex** for lexical analysis to break down SBML files into tokens, **Bison** for creating a parser to validate SBML structure, and **LLVM** for optimizing the generated code. I also applied techniques like loop unrolling and constant folding to improve the efficiency of the generated C++ code.

Problem-Solving and Critical Thinking:

Throughout the project, my **problem-solving skills** evolved significantly. One of the most complex issues I encountered was the translation of biological concepts from SBML into efficient C++ code while maintaining both biological accuracy and computational performance. For example, dealing with complex population dynamics models and rate equations required me to carefully balance computational efficiency and the integrity of the biological model.

5.2 Challenges Encountered and Overcome:

The project posed challenges such as understanding SBML complexity and translating biological models into executable code, while managing different phases like parsing, optimization, and code generation. Overcoming these obstacles helped me develop structured problem-solving, improve time management, and enhance adaptability. This experience fostered resilience and taught me the importance of perseverance, valuable in both personal and professional growth.

Collaboration and Communication:

Although the project was mostly individual, I collaborated with biology experts to ensure the accuracy of the simulation models and validate results. Clear communication with stakeholders helped bridge the gap between computational and biological processes, refining the project based on their feedback. Challenges in aligning terminology and understanding computational limits were overcome through continuous dialogue, improving the project's effectiveness.

5.3**Application of Engineering Standards:**

I followed engineering standards like modular design to ensure the compiler was maintainable and extendable. Prioritizing code quality, I used unit and integration testing to validate the generated C++ code, while adhering to industry practices such as version control and code reviews. Optimization techniques like LLVM were applied to ensure the final product met professional standards.

5.4 Insights into the Industry:

This project highlighted the intersection of computational biology and software engineering, showing how simulation models are applied in drug discovery and biological system modeling. It gave me hands-on experience in creating tools for biological research and simulations, emphasizing the growing demand for bioinformatics tools in the pharmaceutical and biotech sectors. It has solidified my interest in bioinformatics and computational biology, guiding my future career goals. I now aim to work in bioinformatics, focusing on creating real-world simulation tools for biological applications.

5.5 Conclusion of Personal Development:

The capstone project has significantly enhanced my technical skills in compiler design, C++ programming, and biological modeling, while refining my problem-solving and critical thinking abilities. It provided valuable insights into bioinformatics, computational biology, and the importance of interdisciplinary collaboration. This experience has shaped my career goals, equipping me with the skills needed to pursue opportunities in bioinformatics and computational biology.

6. CONCLUSION

The project successfully developed a compiler that translates biological simulation models from SBML (XML) into executable C++ code, using techniques such as lexical analysis, syntax analysis, semantic analysis, optimization, and code generation. It addresses the challenge of efficiently converting biological models into actionable simulations. The solution enhances simulation performance and provides a scalable method for complex biological systems. This compiler significantly improves the ease and speed of simulation

tasks. Ultimately, it offers valuable contributions to computational biology by streamlining the simulation process for researchers.

7. REFERENCE:

1. Smith, J. D. (2018). Computational biology: A practical introduction to bioinformatics and programming. Wiley-Blackwell.
2. Grosu, R., & Nikolai, D. (2019). Biological modeling and simulation: From population dynamics to genomics. Journal of Computational Biology, 26(6), 578–591. https://doi.org/10.1089/cmb.2018.0085
3. SBML Team. (2023). The Systems Biology Markup Language (SBML) specification.
4. Knuth, D. E. (2011). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.
5. Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). *Compilers: Principles, techniques, and tools* (2nd ed.). Addison-Wesley.
6. Mount, D. W. (2004). *Bioinformatics: Sequence and genome analysis*. Cold Spring Harbor Laboratory Press.
7. Wünschiers, R. (2017). *Computational biology: A practical introduction to bio-data processing and analysis with Linux, MySQL, and R*. Wiley-Blackwell.

APPENDICES

Appendix A: Code Snippets

```
#include <stdio.h>

#include <math.h>  // For using the exp() function


// Function to calculate population at time t

double calculate_population(double P0, double r, double t) {

    return P0 * exp(r * t);

}
```

```
int main() {
    double P0 = 1000.0;  // Initial population
    double r = 0.05;     // Growth rate (5% per time unit)
    double t;            // Time variable
    int steps = 20;      // Number of time steps to simulate

    printf("Time\tPopulation\n");

    // Loop over each time step and print population
    for (t = 0; t <= steps; t++) {
        double population = calculate_population(P0, r, t);
        printf("%.2f\t%.0f\n", t, population);
    }

    return 0;
}
```

main.c

Share Run

```c
#include <stdio.h>
#include <math.h>  // For using the exp() function

// Function to calculate population at time t
double calculate_population(double P0, double r, double t) {
    return P0 * exp(r * t);
}

int main() {
    double P0 = 1000.0;  // Initial population
    double r = 0.05;     // Growth rate (5% per time unit)
    double t;            // Time variable
    int steps = 20;      // Number of time steps to simulate

    printf("Time\tPopulation\n");

    // Loop over each time step and print population
    for (t = 0; t <= steps; t++) {
        double population = calculate_population(P0, r, t);
        printf("%.2f\t%.0f\n", t, population);
    }

    return 0;
}
```

Output

```
Time    Population
0.00    1000
1.00    1051
2.00    1105
3.00    1162
4.00    1221
5.00    1284
6.00    1350
7.00    1419
8.00    1492
9.00    1568
10.00   1649
11.00   1733
12.00   1822
13.00   1916
14.00   2014
15.00   2117
16.00   2226
17.00   2340
18.00   2460
19.00   2586
20.00   2718
```

Capstone Project Evaluation Rubric

Total Marks: 100%

Criteria	Weight	Excellent (4)	Good (3)	Satisfactory (2)	Needs Improvement (1)
Understanding of Problem	25%	Comprehensive understanding of the problem.	Good understanding with minor gaps.	Basic understanding, some important details missing.	Lacks understanding of the problem.
Analysis & Application	30%	Insightful and deep analysis with relevant theories.	Good analysis, but may lack depth.	Limited analysis; superficial application.	Minimal analysis; no theory application.
Solutions & Recommendations	20%	Practical, well-justified, and innovative.	Practical but lacks full justification.	Basic solutions with weak justification.	Inappropriate or unjustified solutions.
Organization & Clarity	15%	Well-organized, clear, and coherent.	Generally clear, but some organization issues.	Inconsistent organization, unclear in parts.	Disorganized; unclear or confusing writing.
Use of Evidence	5%	Effectively uses case-specific and external evidence.	Adequate use of evidence, but limited external sources.	Limited evidence use; mostly case details.	Lacks evidence to support statements.
Use of Engineering Standards	5%	Thorough and accurate use of standards.	Adequate use with minor gaps.	Limited or ineffective use of standards.	No use or incorrect application of standards.