COMP2099: Prompt Engineering

Module Specification

University of Nottingham
School of Computer Science
Module Specification: Prompt Engineering
Module Code: COMP2099
Level: Year 2 Undergraduate
Credits: 10
Term: One
Contact Hours: 2 hours lecture + 2 hours lab per week, over 10 weeks
Module Leader: [Module Convenor(s)]

---

1. Module Aims

This module introduces students to the emerging practice of prompt engineering, which involves the design, optimisation, and critical evaluation of interactions with large language models (LLMs) and generative AI systems. It positions prompt engineering as a complement to traditional programming paradigms (Java, C++, Python), emphasising language-as-instruction alongside code-as-instruction. Students will develop both practical skills and critical perspectives, preparing them for applied AI practice in software development, research, and industry contexts.

---

2. Learning Outcomes

On successful completion of this module, students will be able to:

1. **Explain** the principles and limitations of prompt engineering in the context of generative AI systems.
2. **Design and optimise** prompts for diverse tasks, evaluating effectiveness and reliability.
3. **Implement** programmatic prompt workflows and pipelines that integrate with conventional programming.
4. **Analyse and critique** outputs in terms of reproducibility, ethics, bias, and user experience.
5. **Apply** prompt engineering techniques to domain-specific problems in coding, data analysis, and creative applications.
6. **Reflect** on the relationship between prompt engineering and traditional programming, and its role in future computing paradigms.

---

3. Prerequisites

- Successful completion of Year 1 programming modules (Java, C++).
- Familiarity with data structures, algorithms, and software engineering practices.
- Basic Python recommended (for API integration labs).

---

4. Module Outline (Week-by-Week)

Week 1: Introduction to Prompt Engineering

- From procedural and object-oriented programming to natural language programming.
- First explorations with prompting.

Week 2: Anatomy of a Prompt

- Prompt structure, specificity, context setting.
- Zero-shot vs few-shot learning.

Week 3: Reasoning and Decomposition

- Chain-of-thought prompting.
- Multi-step reasoning.

Week 4: Controlling Outputs

- Output constraints, style, persona, templates.

Week 5: Evaluation and Debugging

- Reliability, reproducibility, and prompt testing.

Week 6: Advanced Techniques

- Prompt chaining, tool use, programmatic prompting.

Week 7: Ethics, Bias, and Risks

- Bias, misinformation, responsible use.

Week 8: Human–AI Collaboration

- Prompt engineering as co-programming.
- Usability and user experience design.

Week 9: Prompt Engineering in Practice

- Case studies in software engineering, data analysis, education, creative domains.

Week 10: Project Development and Reflection

- Integration into traditional workflows.
- Peer review and final preparation.

---

5. Teaching and Learning Methods

- **Lectures (20 hours):** Introduce concepts, frameworks, and case studies.
- **Labs (20 hours):** Hands-on practice with LLMs, APIs, and prompt pipelines.
- **Independent Study (90 hours):** Reading, experimentation, and coursework preparation.

---

6. Assessment

- **Coursework (70%)**:
  - *Project*: Design, implement, and critically evaluate a domain-specific prompt engineering application.
  - Deliverables include: code, prompts, evaluation logs, and a reflective report (3,000 words).
- **Written Examination (30%)**:
  - 2-hour exam assessing theoretical understanding, critique of prompts, and ethical reasoning.

---

7. Indicative Reading and Resources

- Bommasani et al. (2021). *On the Opportunities and Risks of Foundation Models*.
- Liu et al. (2023). *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing*.
- OpenAI, Anthropic, Hugging Face API documentation.
- Selected journal articles and case studies provided weekly.

---

8. Graduate Attributes Supported

- **Knowledge and Understanding:** Emerging AI paradigms, language-based programming.
- **Intellectual Skills:** Critical thinking, analytical evaluation of AI outputs.

- **Practical Skills:** LLM experimentation, API integration, workflow design.
- **Transferable Skills:** Communication, collaboration, ethical reasoning.

---

9. Quality Assurance and Fit with Programme

This module builds on programming and software engineering foundations in Year 1. It complements modules in Artificial Intelligence, Human–Computer Interaction, and Data Science. It prepares students for advanced Year 3 options (e.g. NLP, Applied Machine Learning) and research-led projects. The module aligns with UCL's educational commitment to innovation, interdisciplinarity, and responsible digital futures.

Module Outline

Module Description

Prompt Engineering introduces students to the design, optimisation, and critical evaluation of interactions with large language models and generative AI systems. Unlike traditional programming in Java or C++ where precision and syntax dominate, prompt engineering blends computational thinking with linguistic nuance and human–computer interaction. Students will learn techniques for structured prompting, multi-step reasoning, and evaluation of model outputs, while critically reflecting on limitations, ethics, and bias. This module builds on programming fundamentals by shifting from code-as-instruction to language-as-instruction, preparing students for applied AI practice in software development, research, and industry contexts.

Teaching Outline

- **Format**: 2 hours lectures + 2 hours labs weekly
- **Assessment**: Coursework (70%, project-based prompt design and evaluation), Exam (30%, written questions on theory and critique)
- **Pre-requisites**: Completion of Year 1 programming modules (Java/C++) and software engineering concepts
- **Position in Programme**: Complements traditional coding modules by teaching skills in orchestrating generative systems as software components

Weekly Breakdown

Week 1: Introduction to Prompt Engineering

- History of programming paradigms
- Transition from procedural/OO programming (Java, C++) to language-driven programming
- Labs: First experiments with prompting

Week 2: Anatomy of a Prompt

- Structure, specificity, context setting
- Few-shot vs zero-shot examples
- Labs: Comparative prompting tasks

Week 3: Reasoning and Decomposition

- Chain-of-thought prompting
- Stepwise task decomposition
- Labs: Multi-step problem solving with prompts

Week 4: Controlling Outputs

- Style, tone, persona, and constraints
- Prompt templates and patterns
- Labs: Designing structured outputs (JSON, tables, code snippets)

Week 5: Evaluation and Debugging

- Reliability, reproducibility, and prompt testing
- Labs: Systematic evaluation of prompt effectiveness

Week 6: Advanced Techniques

- Prompt chaining, tool use, programmatic prompting
- Integration with APIs
- Labs: Implementing simple prompt pipelines

Week 7: Ethics, Bias, and Risks

- Bias amplification, misinformation, safety issues
- Labs: Identifying and mitigating biased outputs

Week 8: Human–AI Collaboration

- Prompt engineering as co-programming
- Design for usability and user experience
- Labs: Interactive applications using prompts

Week 9: Prompt Engineering in Practice

- Case studies (code generation, education, creative writing, data analysis)
- Labs: Domain-specific prompt design project

Week 10: Project Development and Reflection

- Integration with traditional programming workflows
- Preparing for assessment submission
- Labs: Final project refinement and peer review

Example Lab Tasks

Week 2: Anatomy of a Prompt

- **Task 1**: Write three prompts to summarise the same short article, each varying in specificity (vague, moderate, detailed). Compare outputs.
- **Task 2**: Experiment with zero-shot versus few-shot prompting for text classification (e.g. sentiment analysis). Document differences.

Week 3: Reasoning and Decomposition

- **Task 1**: Create a prompt that solves a basic arithmetic word problem in one step. Then design a chain-of-thought prompt for the same problem. Compare accuracy.
- **Task 2**: Develop a multi-step prompt that outlines pseudocode for a given algorithm, then generates the corresponding Python code.

Week 4: Controlling Outputs

- **Task 1**: Generate a CV in two different formats (bullet points and structured JSON) from the same text description.
- **Task 2**: Write prompts that force a chatbot to answer in the persona of a teacher versus a software engineer. Evaluate consistency.

Week 5: Evaluation and Debugging

- **Task 1**: Test a single prompt 10 times. Record variability in outputs. Develop modifications to reduce inconsistency.
- **Task 2**: Devise a set of automated unit tests (Python) to check that outputs follow a specific format (e.g. valid JSON).

Week 6: Advanced Techniques

- **Task 1**: Use prompt chaining to generate a problem statement, a step-by-step plan, and a final solution.
- **Task 2**: Connect to an LLM API and programmatically iterate over prompt variations. Log results for analysis.

Week 7: Ethics, Bias, and Risks

- **Task 1**: Design prompts on a socially sensitive topic. Identify where outputs exhibit bias or harmful assumptions.
- **Task 2**: Experiment with mitigation strategies (reframing the prompt, adding context, or post-processing outputs).

Week 9: Prompt Engineering in Practice

- **Task 1**: Build a domain-specific "prompt toolkit" (e.g. for debugging Python code or assisting essay writing).
- **Task 2**: Present the toolkit to peers, with demonstration and rationale for design choices.

These tasks deliberately combine **hands-on experimentation** with **reflection and documentation**, so students not only produce outputs but also **analyse why prompts succeed or fail**.

Coursework: Capstone Project in Prompt Engineering (70%)

Weighting: 70% of final grade
Length: 3,000-word report plus supporting materials (code, logs, prompt library)
Deadline: End of Week 11 (one week after final teaching week)
Submission: Via Moodle (single compressed file containing report and artefacts)

Project Brief

You are required to **design, implement, and critically evaluate** a domain-specific prompt engineering application that demonstrates mastery of concepts taught in the module. Your project must show both **practical design work** and **critical reflection**.

You may choose one of the following domains (or propose your own with supervisor approval):

- **Software Development**: Intelligent code assistant for debugging, documentation, or teaching.
- **Data Analysis**: Natural language to structured query or summarisation tool.
- **Creative Applications**: Generative writing, dialogue systems, or interactive storytelling.
- **Education**: Automated tutor, question generator, or explainer system.
- **Professional/Applied Domain**: Healthcare, law, journalism, or other specialised field.

Deliverables

1. **Prompt Library**
    - A collection of prompts (at least 10), organised by function (baseline, refined, advanced).
    - Documentation of design iterations, with justification for changes.
2. **Implementation**
    - A working system or prototype demonstrating use of prompts.
    - May use Python scripts, API calls, or notebook-based workflows.

3. **Evaluation Logs**
    - Systematic testing of prompts (minimum 20 runs).
    - Quantitative or qualitative analysis of reliability, consistency, and quality.
4. **Critical Report (3,000 words)**
    - **Introduction**: Problem domain, motivation, and objectives.
    - **Design Process**: Prompt structures, refinements, and reasoning.
    - **Evaluation**: Strengths, weaknesses, reproducibility, error cases.
    - **Ethical Considerations**: Bias, risks, and mitigation strategies.
    - **Reflection**: Relation to traditional programming paradigms (Java, C++, Python).

Marking Breakdown

- **Prompt Design and Innovation (25%)**: Creativity, clarity, and sophistication of prompts.
- **Implementation and Technical Integration (20%)**: Functionality, correctness, and use of APIs.
- **Evaluation and Testing (20%)**: Rigour in assessing prompt effectiveness.
- **Critical Analysis and Reflection (20%)**: Depth of discussion, ethical awareness, and contextualisation.
- **Presentation and Report Quality (15%)**: Clarity, structure, academic writing, correct referencing.

Guidance

- Projects should balance **technical rigour** with **critical reflection**.
- Overly trivial applications (e.g. single prompt "toy examples") will not be sufficient.
- Students are encouraged to use Git for version control and to log design iterations.
- Collaboration in discussion is allowed, but all submitted work must be individual.

Example Projects

- A **Python notebook** that takes a natural language request and produces **valid SQL queries**, tested across multiple databases.
- A **prompt-driven debugging tool** that accepts user descriptions of bugs and outputs code fixes with explanations.
- An **educational assistant** that explains Year 1 algorithms in multiple levels of complexity (novice, intermediate, advanced).
- A **creative writing generator** with adjustable style parameters, demonstrated with short stories or poems.

This structure ensures the project pulls together **practical, evaluative, and reflective** strands of the module.

Coursework Marking Rubric

Assessment Categories:

1. Prompt Design and Innovation (25%)
2. Implementation and Technical Integration (20%)
3. Evaluation and Testing (20%)
4. Critical Analysis and Reflection (20%)
5. Presentation and Report Quality (15%)

Fail (<40%)

- Minimal or trivial prompt design (e.g. single unrefined prompt).
- No technical implementation, or only copy–paste experiments.
- Little or no evaluation (anecdotal only, no logs).
- No meaningful analysis or reflection.
- Report poorly structured, informal, or incomplete.

Pass (40–59%)

- Basic prompt iteration (2–3 attempts) with limited justification.
- Simple implementation (manual use of Playground, minimal code).
- Limited evaluation (small set of tests, no systematic logging).
- Reflection is shallow (e.g. "prompting is easier than Java").
- Report understandable but lacks academic style or depth.

Merit (60–69%)

- Clear evidence of iterative prompt refinement with justification.
- Functional implementation using code or API, though basic.
- Some systematic evaluation, but may lack depth or consistency.
- Reflection addresses differences with programming paradigms and some ethical issues.
- Report clear, structured, and with some critical engagement.

Distinction (70–100%)

- Sophisticated prompt design: systematic iteration, advanced techniques (constraints, chaining, persona control).
- Strong technical implementation, well-integrated with APIs, automation, or pipelines.
- Rigorous evaluation: multiple test cases, repeated runs, quantitative or qualitative analysis.
- Critical reflection: strong contextualisation with programming paradigms, ethics, and reproducibility.
- Report professional, well-structured, and academically rigorous with references.

Coursework Submission Example: High Distinction

Module: Prompt Engineering
Student ID: XXXXXXX
Project Title: Natural Language to SQL Assistant

---

1. Introduction

The aim of this project is to design a domain-specific application that converts natural language queries into SQL statements using prompt engineering techniques. This problem was chosen because it illustrates the practical challenge of mapping unstructured text into structured, machine-readable commands.

The project builds on traditional programming skills (e.g. parsing, compiler design) but replaces formal syntax rules with **prompt-driven orchestration** of a large language model (LLM). This application demonstrates the potential of language-as-instruction while also highlighting reproducibility and bias concerns.

---

2. Prompt Library

2.1 Baseline Prompt

You are a helpful assistant. Convert the following request into a SQL query.
Request: {user_input}

2.2 Iteration 1 (Structured Instruction)

You are an expert database engineer. Convert the following request into a syntactically correct SQL query.
- Use only SELECT statements.
- Assume the table is called "students" with columns: name, age, degree, grade.
- Output only the SQL code. Do not include explanations.
Request: {user_input}

2.3 Iteration 2 (Error Mitigation)

You are an expert database engineer. Convert the following request into SQL.
- Use table "students" with columns: name, age, degree, grade.
- Only return valid SQL syntax for PostgreSQL.
- If the request is ambiguous, ask a clarification question instead of guessing.
- Output SQL code inside triple backticks.
Request: {user_input}

2.4 Final Advanced Prompt (Robustness + Constraints)

Role: You are a database translator.
Task: Convert the natural language request into SQL.
Constraints:
1. Use table "students" with schema: name (TEXT), age (INT), degree (TEXT), grade (FLOAT).
2. Support only SELECT, WHERE, ORDER BY, and GROUP BY.
3. Always output SQL inside triple backticks.
4. If the request cannot be answered, return: "INVALID REQUEST".
Request: {user_input}

---

3. Implementation

- **Environment**: Python, OpenAI API, SQLite test database.
- **Workflow**:
    - Input: User enters natural language query.
    - Processing: Prompt is dynamically inserted into advanced template.
    - Output: SQL query returned, executed against SQLite test database.
    - Post-processing: If execution fails, re-prompt with debugging instructions.

Code Excerpt:

```
import openai, sqlite3

def nl_to_sql(user_input):
  prompt = f"""
  Role: You are a database translator.
  Task: Convert the natural language request into SQL.
  Constraints:
  1. Use table "students" with schema: name (TEXT), age (INT), degree (TEXT), grade (FLOAT).
  2. Support only SELECT, WHERE, ORDER BY, and GROUP BY.
  3. Always output SQL inside triple backticks.
```

```
4. If the request cannot be answered, return: "INVALID REQUEST".

Request: {user_input}
"""
response = openai.ChatCompletion.create(
  model="gpt-4",
  messages=[{"role": "user", "content": prompt}]
)
return response["choices"][0]["message"]["content"]
```

---

4. Evaluation

4.1 Test Queries

Natural Language Request	Expected SQL	Result	Notes
"List all students"	SELECT * FROM students;	✅	Accurate
"Show names of students older than 21"	SELECT name FROM students WHERE age > 21;	✅	Correct
"Find average grade per degree"	SELECT degree, AVG(grade) FROM students GROUP BY degree;	✅	Worked reliably
"List students ordered by grade"	SELECT * FROM students ORDER BY grade;	✅	Deterministic
"Delete all students"	INVALID REQUEST	✅	Correct rejection
"Who are the best students?"	Clarification requested	✅	Handled ambiguity appropriately

4.2 Reliability Testing

- **20 runs per query** conducted.
- Baseline prompt produced variable outputs (success rate: 60%).
- Advanced prompt produced consistent outputs (success rate: 95%).
- Failures occurred when user input was vague. Clarification mechanism reduced errors.

---

5. Ethical Considerations

- **Bias**: If trained on biased examples (e.g. stereotypes about degrees), model could return unfair groupings. Mitigated by constraining schema.
- **Safety**: Prohibited destructive SQL queries (DROP, DELETE) via explicit constraints.
- **Transparency**: Outputs formatted clearly; ambiguous cases flagged rather than guessed.

---

6. Reflection

This project illustrates the shift from **code-as-instruction** to **language-as-instruction**. In Java or C++, ambiguity is resolved through strict syntax and compilation errors. In prompt engineering, ambiguity persists unless carefully managed through design. Prompt reproducibility is therefore less reliable than compiled languages.

The iteration process mirrored software debugging: baseline prompts were “buggy”, requiring refinement until outputs stabilised. Unlike C++ or Java, correctness here depends on **guiding a probabilistic system**, not enforcing deterministic rules.

Prompt engineering therefore feels closer to **co-programming with a non-deterministic partner** than traditional software development.

---

7. Conclusion
The project successfully demonstrated a domain-specific prompt engineering system for natural language to SQL translation. The final prompt library achieved 95% reliability across test queries and incorporated safeguards against invalid or unsafe outputs. The work highlights both the opportunities and limitations of prompt-driven systems and provides a foundation for future integration into more robust database tools.

Appendices
Appendix A: Full Prompt Iteration Logs
(included in compressed submission folder)
Appendix B: Python Notebook
(included in compressed submission folder)

This would represent a **high-distinction submission**: strong design, rigorous evaluation, critical reflection, and explicit ethical awareness.

Marker Commentary for Strong Submission (Natural Language to SQL Assistant)

- **Prompt Design and Innovation (20/25)**: Systematic iteration from baseline to advanced prompts. Strong use of constraints (valid SQL, rejection of unsafe queries). Could have explored few-shot methods for breadth.
- **Implementation and Technical Integration (18/20)**: Python–SQLite integration well executed. Post-processing for errors shows maturity. Some automation could enhance robustness.
- **Evaluation and Testing (14/20)**: 20-run tests with reliability metrics demonstrate rigour. Considered edge cases effectively. More quantitative scoring could strengthen analysis.
- **Critical Analysis and Reflection (16/20)**: Excellent comparison with traditional programming and strong insight into co-programming. Ethical reflection present but slightly brief.
- **Presentation and Report Quality (14/15)**: Well structured, professional, and balanced. Clear technical detail with critical depth.

Total: 82/100 → High Distinction

- **General Feedback**: An excellent, rigorous, and reflective submission. Iterative design, evaluation, and contextual analysis make this a model of distinction-level work.

Coursework Submission Example: Pass

Module: Prompt Engineering
Student ID: XXXXXXX
Project Title: Chatbot for Study Advice

---

1. Introduction

This project creates a chatbot that helps students with study advice. The chatbot is based on prompts given to a large language model. The idea is to get the chatbot to act like a tutor and give tips. I wanted to test if prompts can be used to make helpful advice.

---

2. Prompt Library

Prompt 1 (Baseline)

You are a helpful tutor. Give study advice to the student.

Prompt 2 (Refined)

You are a tutor. Give study advice to the student in 3 bullet points.

Prompt 3 (Final)

You are a tutor. Answer the question in short clear sentences. Give three tips maximum.

---

3. Implementation

The chatbot was tested using the OpenAI Playground. I copied the prompts into the interface and got the outputs. I did not integrate the chatbot into any code, but I saved the responses.

Example input:

- User: "How do I revise for exams?"
- Output:
    - "1. Review your notes. 2. Practise past papers. 3. Take breaks."

Example input:

- User: "I am stressed about deadlines."
- Output:
    - "1. Make a plan. 2. Do one task at a time. 3. Rest when needed."

---

4. Evaluation

I tested each prompt with five inputs. The responses were mostly fine. Sometimes they were too long, but overall they worked.

Test queries:

- "How to prepare for lectures?" → Good answer.
- "What is a good way to remember things?" → OK answer.
- "What do I do if I fail a module?" → Answer was vague.

I did not do many runs because the answers were quite similar.

---

5. Ethical Considerations

The chatbot could give wrong advice, but I did not find anything harmful. It is not perfect, but good enough for advice.

---

6. Reflection

Prompt engineering is about making good prompts. It is different from programming in Java or C++ because it is easier to write. I think it is useful but less reliable.

---

7. Conclusion

I showed that prompts can make a chatbot for study advice. It mostly worked and gave helpful tips. I did not have time to do more tests, but I think it shows that prompt engineering is useful.

Marker Commentary

- **Prompt Design (8/25):** Only three simple prompts. No sophistication, no iteration beyond word changes.

- **Implementation (8/20):** No integration with code or APIs. Merely copy–paste into Playground.
- **Evaluation (8/20):** Very limited testing. No reproducibility checks, no structured analysis.
- **Critical Analysis (8/20):** Almost absent. Reflection is superficial ("easier than Java").
- **Presentation (8/15):** Coherent but short and lacking academic style.

Total: 40/100 → Pass

This would represent a **pass** submission: This kind of submission *does enough* to show the student understood the basics, but lacks depth, rigour, and serious engagement.

Exam Paper: (30%)

University of Nottingham
School of Computer Science
Module: Prompt Engineering
Exam Duration: 2 hours

Answer THREE questions. Question 1 is compulsory.
All questions carry equal marks.

Question 1 (Compulsory)
(a) Explain how prompt engineering differs from traditional programming in languages such as Java or C++. Your answer should reference syntax, determinism, and error handling.
(10 marks)
(b) Define the following terms in the context of large language models:

- Zero-shot prompting
- Few-shot prompting
- Chain-of-thought prompting
  (9 marks)

(c) A student writes the following prompt for an LLM:
"Write a program."
The output is inconsistent and not useful. Identify **three reasons** for this failure and **redesign the prompt** to improve output quality.
(11 marks)
[Total: 30 marks]

Question 2
(a) Discuss the concept of **prompt reproducibility**. Why is it harder to achieve compared to reproducibility in traditional compiled programming?
(10 marks)
(b) You are tasked with generating a JSON object containing student data using an LLM. Design a prompt that ensures the output conforms to the JSON format. Explain how you would test and validate the reliability of your prompt.
(20 marks)
[Total: 30 marks]

Question 3
(a) Identify and discuss **two ethical risks** associated with prompt engineering in socially sensitive domains (e.g. healthcare, law, journalism).
(12 marks)
(b) Propose a strategy for **bias mitigation** in LLM outputs. Illustrate with an example prompt before and after applying your strategy.
(18 marks)
[Total: 30 marks]

Question 4
(a) Compare **human–AI co-programming** via prompt engineering with pair programming in traditional software engineering. Highlight both opportunities and limitations.
(12 marks)
(b) Case Study: An educational start-up wants to use prompt engineering to build a tutoring system that explains algorithms to undergraduates.

- Identify two design challenges.
- Suggest concrete prompt engineering techniques to address them.
- Discuss how you would evaluate the effectiveness of the system.
  (18 marks)

[Total: 30 marks]

Marking Guidance

- Demonstrate **clarity of explanation**, not just technical detail.
- Where examples are requested, provide specific prompts or outputs.
- Critical reflection (advantages, limitations, risks) is valued as much as technical accuracy.

This exam covers:

- **Core knowledge** (terminology, definitions).
- **Applied skills** (prompt design, testing).
- **Critical reflection** (ethics, co-programming).

Exam Paper Submission Example: High Distinction

Exam Answers – Prompt Engineering
Student ID: XXXXXXX
Module: Prompt Engineering

Question 1 (Compulsory)
(a)
Prompt engineering differs from programming in Java or C++ in several key ways:

- **Syntax and precision**: In Java or C++, instructions must be syntactically exact. In prompt engineering, natural language is accepted, which allows flexibility but introduces ambiguity.
- **Determinism**: Java and C++ programs execute deterministically. Prompts interact with probabilistic models, so the same input may yield different outputs.

- **Error handling**: Traditional languages give explicit compiler/runtime errors. In prompting, "errors" manifest as irrelevant, inconsistent, or incorrect outputs, requiring rephrasing rather than debugging syntax.

(9/10 – excellent but could have given one more concrete coding example.)

(b)

- **Zero-shot prompting**: Asking the model to complete a task without any examples.
- **Few-shot prompting**: Providing several examples within the prompt to guide the model.
- **Chain-of-thought prompting**: Encouraging the model to reason step by step, often by explicitly instructing it to "think through" the problem before producing an answer.

(9/9 – precise definitions.)

(c)

Reasons for failure:

1. The request "Write a program" is too vague — no specification of language, purpose, or constraints.
2. The model defaults to arbitrary interpretations (e.g. Python vs Java) leading to inconsistency.
3. Lack of formatting guidance, so output may include explanations, comments, or partial code.

Improved prompt:

You are an expert Java developer. Write a complete Java program that prints "Hello, World!".
Constraints:
- Provide only code, no explanations.
- The program must compile and run without modification.

(9/11 – strong but could have noted reproducibility issues more explicitly.)

Total Q1: 27/30

Question 2

(a)

Prompt reproducibility means getting consistent outputs from the same prompt across runs. It is difficult because LLMs are non-deterministic; they sample from probability distributions. In contrast, compiled Java/C++ programs yield identical results unless code or environment changes. Prompt reproducibility is affected by randomness, context length, and subtle variations in wording.

(8/10 – clear but could mention temperature/top-k parameters.)

(b)

Prompt Design:

You are an assistant that outputs data strictly in JSON format.
Task: Convert the following request into a valid SQL-like query result expressed as JSON.
Rules:
- Only use the schema: { "name": TEXT, "age": INT, "degree": TEXT, "grade": FLOAT }.
- Output must be valid JSON, no explanations.
Request: {user_input}

Testing and Validation:

- Run the prompt 20 times per query, ensuring JSON parses correctly each time.
- Use Python's json.loads() to automatically validate outputs.

- Measure reliability as percentage of runs producing valid JSON.
- Introduce deliberately ambiguous inputs to see if the model handles them gracefully.

(17/20 – strong, practical, but could have shown a concrete input/output example for completeness.)

Total Q2: 25/30

Question 3

(a)

Two ethical risks:

1. **Bias amplification**: For example, a healthcare prompt might assume default patients are male, leading to unequal advice.
2. **Misinformation**: In journalism, prompts could generate false or misleading "facts", risking reputational harm.

(10/12 – strong examples, could briefly mention accountability.)

(b)

Strategy: Prompt reframing with explicit fairness constraints.

Before (biased):

"Write a profile of a typical software engineer."

- Likely biased towards male, young, Western stereotypes.

After (mitigated):

"Write three diverse profiles of software engineers, varying in gender, age, and background. Ensure accuracy and inclusivity."

(15/18 – good strategy, clear illustration, but could have noted post-processing checks.)

Total Q3: 25/30

Question 4

N/A

Final Mark Breakdown

- Q1: 27/30
- Q2: 25/30
- Q3: 25/30

Total: 77/90 → Distinction

Exam Paper Example: Pass

Exam Answers – Prompt Engineering
Student ID: XXXXXXX
Module: Prompt Engineering

Question 1 (Compulsory)

(a)

Prompt engineering is not like Java or C++ because you can just type what you want in English. Java and C++ are strict and you need semicolons and things. Prompting is easier but sometimes the answers are random.

(5/10 – mentions syntax and randomness but lacks detail on error handling or determinism.)

(b)

- Zero-shot prompting: the model gets no examples.
- Few-shot prompting: the model gets some examples.
- Chain-of-thought prompting: the model writes down the steps.

(6/9 – basic but correct, limited explanation.)

(c)

The prompt "Write a program" does not work because it is too short. The model does not know which language or what to do.

Better prompt:

"Write a Python program that prints Hello World."

(5/11 – identifies vagueness but improved prompt still very minimal.)

Total Q1: 16/30

Question 2

(a)

Reproducibility means the same output every time. In programming languages this is guaranteed, but prompts can give different answers each time because the AI is not exact.

(6/10 – captures the idea but lacks explanation of probability, randomness, or parameters.)

(b)

Prompt:

Give me student data in JSON.

Testing: I would try it a few times and check if the JSON looks correct.

(4/20 – extremely weak; no schema, constraints, or validation method.)

Total Q2: 10/30

Question 3

N/A

Question 4

(a)

Human–AI co-programming means you work with AI, but pair programming is with another human. The AI can give answers quicker, but sometimes wrong. Pair programming is more reliable.

(6/12 – very limited, but at least some comparison.)

(b)

Case study: For a tutoring system, one problem is the AI may make mistakes. Another is it might be confusing.

Technique: Use prompts like "Explain simply in short steps."

Evaluation: Ask students if it helped.

(8/18 – basic but relevant, lacks depth or rigour.)

Total Q4: 14/30

Final Mark Breakdown

- Q1: 16/30
- Q2: 10/30
- Q4: 14/30

Total: 36/90 → Pass