Prompt Engineering for Software Developers

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

What is Prompt Engineering?

"Prompt engineering is the process of designing high-quality prompts that guide LLMs to produce accurate outputs."

Lee Boonstra, Google

"Prompt engineering is the process of designing, refining, and optimizing input prompts to effectively communicate the user's intent to a language model."

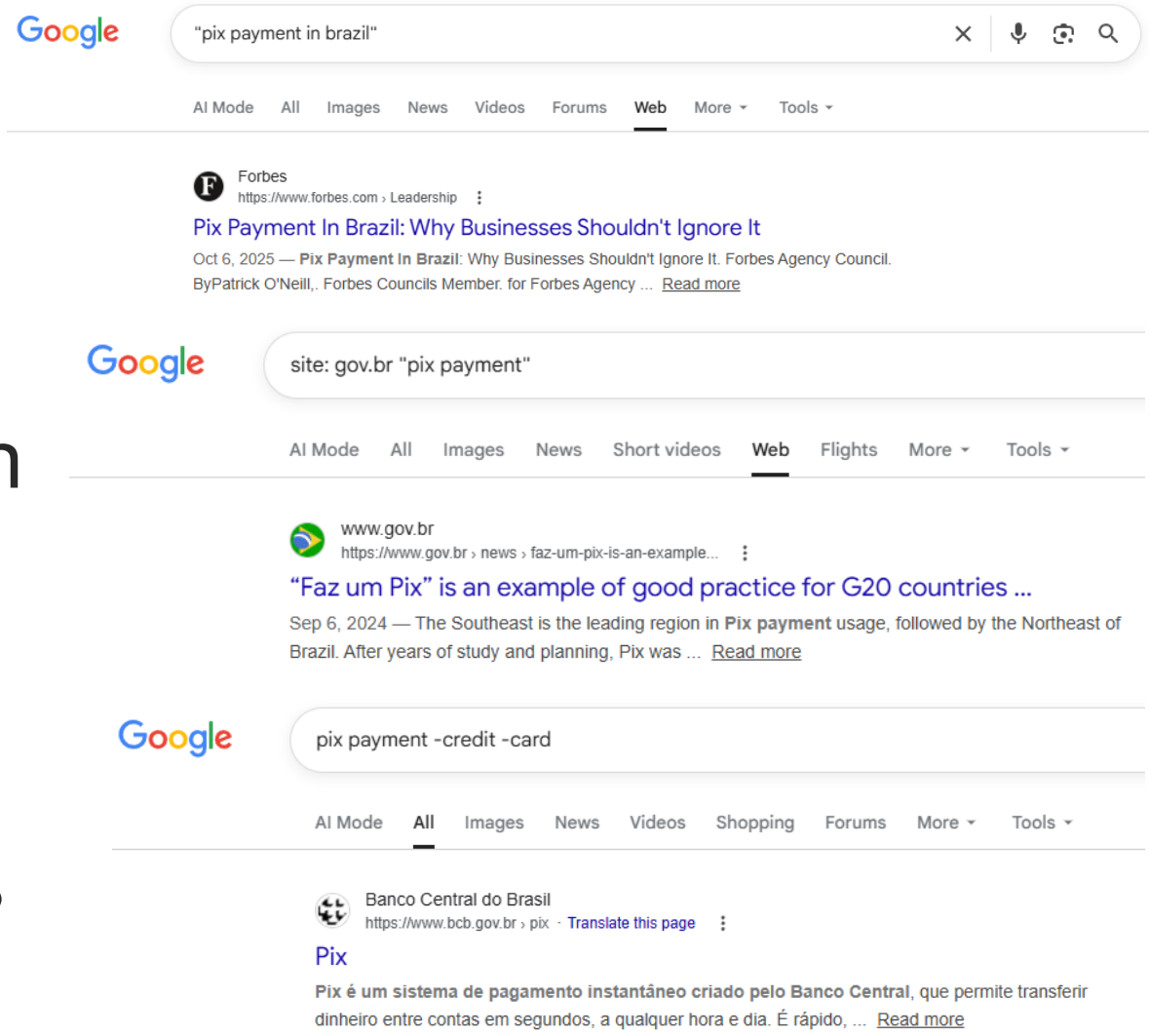
Sabit Ekin, 2023

Google Prompting

- Is this a good way of searching Google?
  - Why?
  - Why not?

Effective Google Seach

- Search for exact terms
- Search within certain sites
- Exclude “noisy” terms



ChatGPT Prompting

Prompt: Java implementation of factorial

Output:

```
public class Factorial {
  public static long factorial(int n) {
    if (n < 0) {
      throw new IllegalArgumentException("Number must be non-negative");
    }
    long result = 1;
    for (int i = 2; i <= n; i++) {
      result *= i;
    }
    return result;
  }
}
```

- Is this a good ChatGPT prompt? Why?

Basics of Prompt Engineering

- Communicate the user intent
- Aim for the model understanding
- Give information about the domain
- Specify constraints

Predicting the Next Token

- Given a sequence of tokens, the model predicts the most likely next token

I love visiting _____

0.40: places

0.15: new

0.10: beautiful

Models do not always pick the top token.

I love visiting places _____

0.20: where

0.17: that

0.12: and

Models do not always pick the top token.

Prompt and Outputs

I love visiting

new places, exploring different cultures.

places that make me feel inspired and alive.

beautiful places and creating unforgettable memories.

places where I can relax, explore, and learn something new.

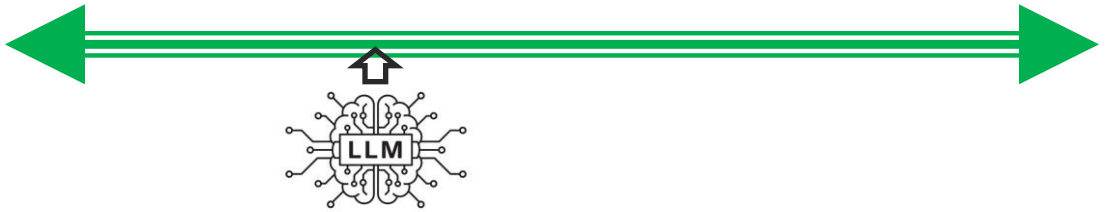
Prompt

GenAI Outputs

Model Temperature

- Temperature controls the randomness or creativity of the large language model

more deterministic



more random

I love visiting

0.40: places

0.15: new

0.10: beautiful

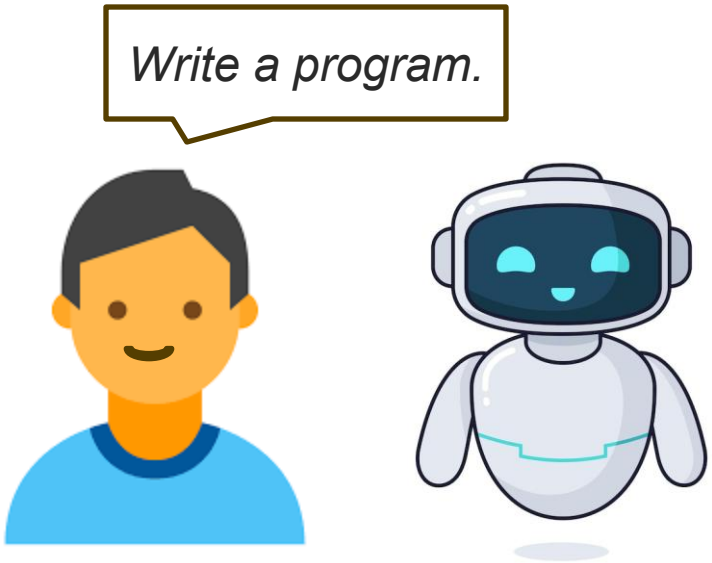
I love visiting

0.05: deep

0.02: a galaxy

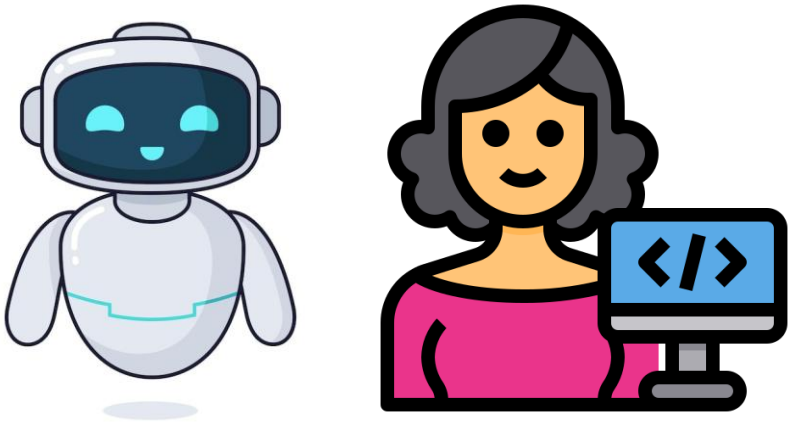
0.01: centuries

An Example of Bad Prompt



An Example of Good Prompt

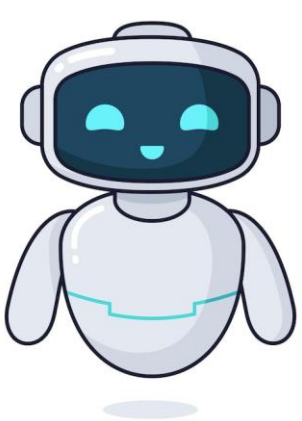
Write a program in Java to calculate the factorial of an integer number. The program should have one class named FactorialExample and one method called factorial. Use recursion.



Output of a Good Prompt

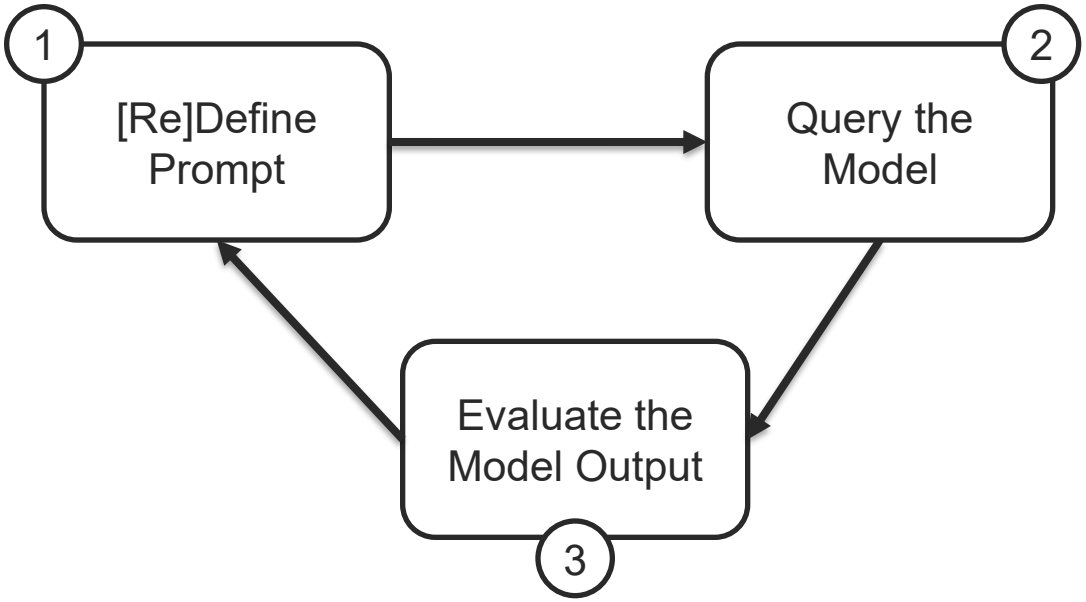
```java
public class FactorialExample {

  public static long factorial(int n) {
    if (n == 0)
      return 1; // base case
    return n * factorial(n - 1);
  }
}
```

Write a program in Java to calculate the factorial of an integer number. The program should have one class named FactorialExample and one method called factorial. Use recursion.



Iterative Process

- Creating a good prompt is often an iterative process



Guidelines of a Good Prompt

- A good prompt should have four key elements
  - *Clarity*: write clear instructions
  - *Specificity*: write specific instructions
  - *Relevance*: provide only relevant information
  - *Context*: give context to the model

Clear Instructions

- Define the model role in the prompt
- Use delimiters to separate different parts of the prompt
  - Triple quotes: """
  - XML tags: <tag> </tag>
  - All caps: EXAMPLE
- Ask for a structured output (e.g., JSON)
  - It makes easier to parse the output
  - You can integrate into your pipeline

Prompt Delimiters (''')

```
You are an expert Java developer.
---
Which code below is better in terms of readability?

'''
public class ReverseString {
  public static String reverse(String input) {
    return new StringBuilder(input).reverse().toString();
  }
}
'''
public class ReverseManual {
  public static String reverse(String input) {
    String reversed = "";
    char[] chrs = input.toCharArray();
    for (int i = chrs.length - 1; i >= 0; i--) reversed += chrs[i];
    return reversed;
  }
}
'''
```

Prompt Delimiters <tag></tag>

```
You are an expert Java developer.
Which code below is better in terms of readability?

<code1>
public class ReverseString {
  public static String reverse(String input) {
    return new StringBuilder(input).reverse().toString();
  }
}
</code1>
<code2>
public class ReverseManual {
  public static String reverse(String input) {
    String reversed = "";
    char[] chrs = input.toCharArray();
    for (int i = chrs.length - 1; i >= 0; i--) reversed += chrs[i];
    return reversed;
  }
}
</code2>
```

Specify Constraints

- Ask the model to check if conditions are satisfied
  - It helps the model to identify invalid data
- Specify steps to break down complex tasks
  - Complex tasks should be broken down
- Let the model work out its own solution before jumping into conclusions

JSON and Constraints

```
Generate a list of methods from the following Java code.

Provide this list in JSON format with the following keys:
ClassName, MethodName, ReturnType.

<code>
public class FastCar {

  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
</code>

If the code does not contain a method, then write "No method found".
```

Give Context

- It enables in-context learning
  - The model understands the desired format, style, or task without additional training.
- Common types of in-context prompting
  - *Zero-shot*: No examples are provided.
  - *One-shot*: A single example is provided.
  - *Few-shot*: Multiple examples are provided.
  - *Chain-of-Thought (CoT)*: Examples include the reasoning steps to guide the model.

Zero-Shot Prompting

- A zero-shot prompt is the simplest type of prompt
  - It provides a description of a task without examples

Prompt #1

I need to check if the Java code below contains code smells (aka bad smells). Could you please identify which smells occur in the following code? However, do not describe the smells, just list them.

Please start your answer with "YES I found bad smells" when you find any bad smell. Otherwise, start your answer with "NO, I did not find any bad smell".

When you start to list the detected bad smells, always put in your answer "the bad smells are:" amongst the text your answer and always separate it in this format: 1. Long method, 2. Feature envy

[Java source code with the smell]

L. Amorim *et al*. "**Bad Smell Detection using Google Gemini**". AIML, 2025.

Few-Shot Prompting

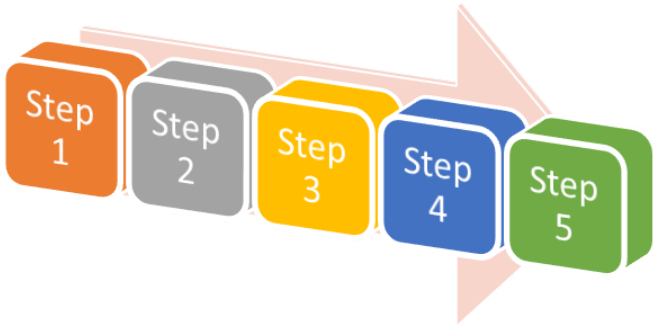
- When creating prompts for AI models, it is often helpful to provide examples.
  - Give examples of successful completion
  - Then ask the model to perform the task
- Examples are useful when you want the model to follow a certain output structure or pattern.

Example of One-Shot Prompt

```
Generate a list of methods from the given Java code. Provide this
list in JSON format with the following keys: ClassName, MethodName.

<example>
public class FastCar {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }
}

'''
[ { "ClassName": "FastCar", "MethodName": "fullThrottle" } ]
'''
</example>

<code>
  ...
</code>
```

Chain of Thoughts (CoT)

- Give a set of steps to complete the task
  - CoT is very useful for tasks that require reasoning, such as long computation



- You can also combine CoT with a few-shot prompting to get better results

Example of a CoT Prompt

Prompt

You are a software expert analyzing one Java file for symptoms that may indicate the `[Smell name]` code smell.

`[Description of the smell]`

Please answer the following questions step by step before reaching a conclusion about the code smell.

`[Chain of thoughts with steps to detect each specific smell]`

Instructions:
Based on the step-by-step questions above, you should indicate if the following Java file has the code smell. Please start your answer with "YES, I found `[Smell name]`" if you detect symptoms that could indicate this smell, or "NO, I did not find `[Smell name]`" if you do not. Briefly explain your reasoning, answer the questions and provide the summary.

`[Java source code with the smell]`

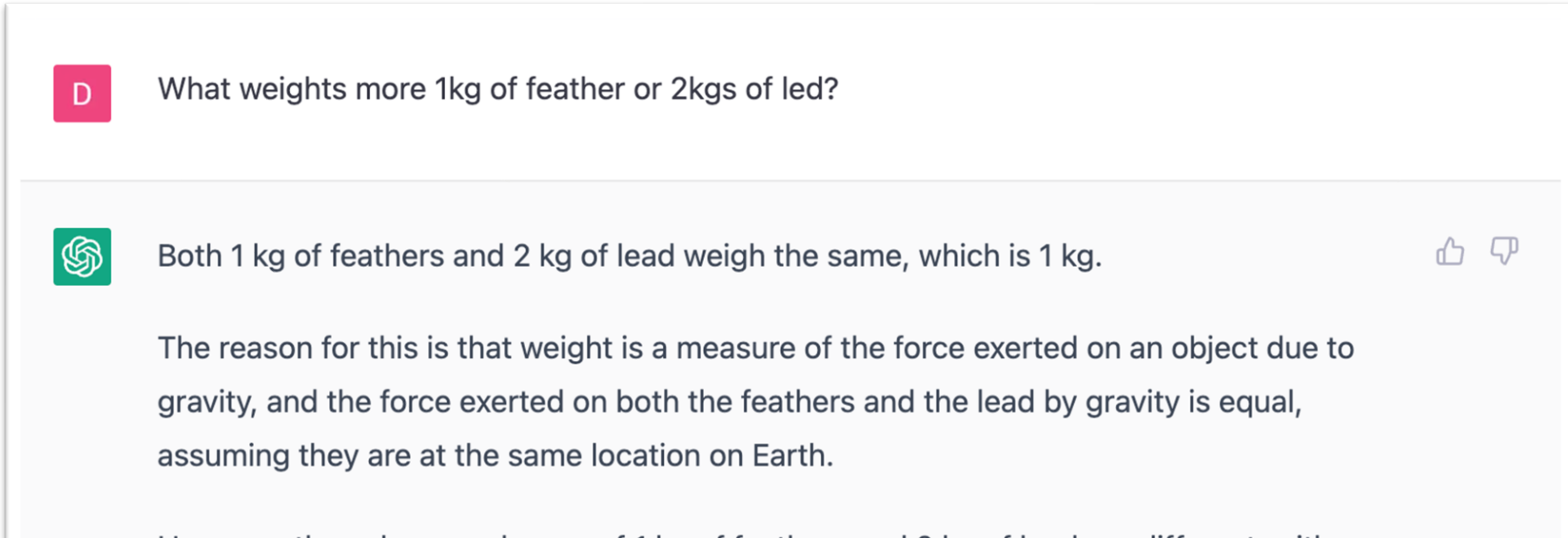
Steps for God Class

1. Verify if the class centralize the system intelligence
2. Inspect if the class has non-cohesive sets of methods
3. Check if the class is large and complex

S. Souza *et al*. "**Assessing the Effectiveness of Large Language Models for Code Smell Detection**". Preprint, 2026.

Hallucination

- LLMs are great at making stuff up
  - They give an answer that does not make sense

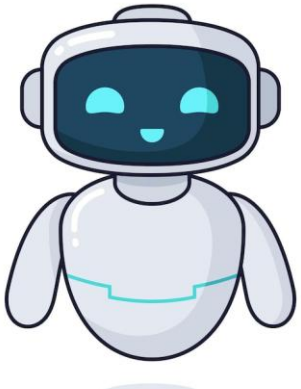


Example of Code Hallucination

According to SonarQube, we should “Track Uses of TODO Tags”. Can you address this issue in the following code?

```
'''
else {
// TODO(cushon): don't break after here
// for consistency with for, while, etc.
  builder.breakToFill();
  builder.open(ZERO);
  scan(ex, null);
  builder.close();
}
```

H. Nunes *et al*. "**Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-World Projects**". SANER, 2025.



```
else {
// TODO(cushon): don't break after here
// for consistency with for, while, etc.
  builder.breakToFill();
  builder.open(ZERO);
  scan(ex, null);
  builder.close();
}

// Track uses of TODO tags
if (node.getBlock().toString().contains("TODO")) {
  // Perform tracking actions here
}
```

Software Engineers Need

The World Needs More Software Engineers

A Conversation with Box CEO Aaron Levie

By Tim O'Reilly
April 7, 2026 • 10 minute read

Share

I sat down with Aaron Levie at the O'Reilly AI Codecon two weeks ago. Aaron cofounded Box in 2005, and 20 years later, his company manages content for about two-thirds of the Fortune 500. Aaron is one of the few CEOs of an incumbent enterprise software company thinking deeply in public about what AI means for the entire enterprise stack. There are a lot of people who are building companies from the ground up with AI, others who are dragging their feet adapting existing enterprises to it, and then there's Aaron. He sits in a kind of Goldilocks zone, enthusiastic but not uncritical, engaging in the hard work of adapting AI to the enterprise and the enterprise to AI.

Relevant Research Paper

Large Language Models for Software Engineering: A Systematic Literature Review

XINYI HOU and YANJIE ZHAO, Huazhong University of Science and Technology, Wuhan, China
YUE LIU, Monash University, Melbourne, Australia
ZHOU YANG, Singapore Management University, Singapore
KAILONG WANG, Huazhong University of Science and Technology, Wuhan, China
LI LI, Beihang University, Beijing, China
XIAPU LUO, The Hong Kong Polytechnic University, Hong Kong, China
DAVID LO, Singapore Management University, Singapore
JOHN GRUNDY, Monash University, Melbourne, Australia
HAOYU WANG, Huazhong University of Science and Technology, Wuhan, China

Literature review of 395 papers that use LLMs in SE

Large Language Models (LLMs) have significantly impacted numerous domains, including Software Engineering (SE). Many recent publications have explored LLMs applied to various SE tasks. Nevertheless, a comprehensive understanding of the application, effects, and possible limitations of LLMs on SE is still in its early stages. To bridge this gap, we conducted a Systematic Literature Review (SLR) on LLM4SE, with a particular focus on understanding how LLMs can be exploited to optimize processes and outcomes. We selected and analyzed 395 research articles from January 2017 to January 2024 to answer four key Research Questions (RQs). In RQ1, we categorize different LLMs that have been employed in SE tasks, characterizing their distinctive features and uses. In RQ2, we analyze the methods used in data collection, pre-processing,

X. Hou *et al*. "**Large Language Models for Software Engineering: A Systematic Literature Review**". TOSEM, 2024.

Bibliography

- Diego Costa. **Generative Artificial Intelligence for Software Engineering**. Concordia University, 2025.

- Isa Fulford, Andrew Ng. **ChatGPT Prompt Engineering for Developers**. DeepLearning.AI, 2025.