

Code Summarization (Java)

Prompt method:	Prompt:	GPT-4 output:	Gemini output:
Zero-shot	Summarize the functionality of the following method: [code can be found in RawData.pdf]	The <code>countWordFrequency</code> method takes a list of strings (<code>words</code>) and returns a map (<code>HashMap</code>) that represents the frequency of each word in the list.	This method, <code>countWordFrequency</code> , takes a list of strings as input and returns a map where each unique string from the input list is a key, and its corresponding value is the number of times that string appears in the list. Essentially, it counts the frequency of each word in the provided list.
Few-shot	Example 1: <pre>def is_even(n): if n % 2 == 0: return True else: return False</pre> <p>This method determines whether a number is even.</p> <p>Example 2: <pre>def is_prime(n): if n <= 1: return False for i in range(2, n): if n % i == 0: return False return True</pre><p>This method determines if a number is prime.</p><p>Summarize the functionality of the following method: [code can be found in RawData.pdf]</p></p>	This method counts how many times each word appears in a list.	This method calculates and returns the frequency of each unique word within a given list of words. It does this by iterating through the list and using a map to store each word as a key, with its corresponding count as the value.

This task involved providing a summary of a Java method. Using zero-shot resulted in a more lengthy, specific output which summarizes the function very well. Using few-shot resulted in shorter outputs, using more casual language and hitting less keywords. This is likely due to the fact that the examples given use more casual language, so the prompt guides the model to respond in a similar way. Chat-GPT gave shorter, more concise outputs, summarizing the function using more commonplace language. Gemini's approach was more code-based, which is much more apparent when comparing the few-shot outputs from each model.

Bug Fixing (Python – Off-by-One)

Prompt-chaining	Tell me where the off-by-one error is in this function (do not fix it): [code can be found in RawData.pdf] -> (Prompt for GPT) fix the off-by-one error in this line: <pre>for i in range(start, end):</pre> (Prompt for Gemini) fix the off-by-one error in the for loop condition.	The off-by-one error is in this line: <pre>for i in range(start, end):</pre> This line causes the loop to exclude <code>end</code> , because <code>range()</code> in Python goes from <code>start</code> up to but not including <code>end</code> . If the intention is to include <code>end</code> in the sum, this creates an off-by-one error by leaving out the last value. -> To fix the off-by-one error in this line: <pre>for i in range(start, end):</pre> You should change it to: <pre>for i in range(start, end+1):</pre>	The off-by-one error is in the <code>for</code> loop condition: <code>range(start, end)</code> . This will iterate from <code>start</code> up to, but not including, the value of <code>end</code> . -> <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre> The <code>range</code> function now goes up to <code>end + 1</code> , ensuring that the loop includes the <code>end</code> value in its iteration.
Self-consistency	Run several examples to determine why there is an off-by-one error in this code. Then, find the most efficient way to fix it: [code can be found in RawData.pdf]	The <code>range(start, end)</code> function does not include the <code>end</code> value, so you're always missing the last number. <pre>def sum_range(start, end): total = 0 for i in range(start, end + 1): total += i return total</pre>	The problem lies in how the <code>range()</code> function in Python works. <code>range(start, end)</code> generates a sequence of numbers from <code>start</code> (inclusive) up to, but not including, <code>end</code> . Most Efficient Fix <pre>def sum_range_corrected(start, end): return sum(range(start, end + 1))</pre>

This task involved locating an error in a Python function, and finding a solution. Using prompt-chaining resulted in a very clear, direct output from both models. This could be because by chaining the prompts together, the model follows a guided "train of thought". Self-consistency led to a similar output from Chat-GPT, but Gemini provided a different response. This could be because the model was specifically asked to explore several options when finding a solution, and to choose the most efficient one. It is interesting that Chat-GPT does not provide the same solution, but instead chose the solution that preserves most of the code.

Bug Classification (C++)

Chain-of-thought	Classify the type of bug in the following C++ function. No need to provide a solution. Do it step by step : [code can be found in RawData.pdf]	Final Bug Type: Dangling Pointer / Returning a Pointer to a Local Variable. This is a common issue where a pointer to a local variable is returned, leading to memory that is no longer valid once the function scope ends.	Returning a dangling pointer is a significant memory management error . Attempting to dereference this pointer in the calling code leads to undefined behavior , which can result in program crashes, data corruption, or other unpredictable issues. Therefore, the type of bug in this C++ function is a memory management error specifically resulting in a dangling pointer due to returning a pointer to a local variable.
Self-consistency	What type of bug is in this C++ function? Run it on a few examples and give the most common bug experienced. [code can be found in RawData.pdf]	Type of Bug: The type of bug in this C++ function is dangling pointer . This happens because the function returns a pointer to a local array that exists only within the function's scope. Once the function exits, the memory used by the array is deallocated, making the returned pointer point to invalid memory. Common Bug Experienced: <ul style="list-style-type: none">The most common bug encountered with this type of code is accessing invalid memory after the function call. Since the array is local to the function and is deallocated when the function exits, any attempt to use the returned pointer results in accessing memory that is no longer valid. This often leads to segmentation faults, corrupted data, or undefined behavior.	The type of bug in this C++ function is a memory management error , specifically resulting in a dangling pointer . Let's consider what happens when this function is used. Most Common Bug Experienced: The most common bug experienced when using this function is reading garbage values or incorrect data when trying to access the array elements through the returned pointer in the calling function. This is because the memory the pointer points to is no longer guaranteed to hold the original data. In more severe cases, it can lead to program crashes due to accessing memory that is no longer considered valid for the program to read. While the first example might coincidentally "work," it's crucial to understand that this is undefined behavior and highly unreliable. The second example demonstrates the more typical and problematic outcome.

This task involved classifying a type of bug located within a C++ function. With chain-of-thought prompting, the Chat-GPT was concise and had a more casual, conversational output. Gemini used more technical terms, and explained some additional concepts of what occurs in the error. The self-consistency prompt led to longer outputs from both models, and both outputs contained technical terms and a more well-rounded response about the outcomes of the bug.

Generating Email Validators (Python + Regex)

Zero-shot	Complete the function using regex to validate basic email addresses: [code can be found in RawData.pdf]	<pre>import re def is_valid_email(email): pattern = r"^[a-zA-Z0-9_%.+]*@[a-zA-Z0-9-]*\.[a-zA-Z]{2,}\$" return re.fullmatch(pattern, email) is not None</pre>	<pre>import re def is_valid_email(email): """Basic email validation using a simple regex""" return bool(re.match(r"^[^@]+@^[^@]+\.[^@]+\$", email))</pre>
Self-consistency	Complete the function in several ways using regex to validate basic email addresses, and return the most efficient. [code can be found in RawData.pdf]	<pre>import re def is_valid_email(email): return re.fullmatch(r"[a-zA-Z0-9_%.+]*@[a-zA-Z0-9-]*\.[a-zA-Z]{2,}", email) is not None</pre>	<pre>def is_valid_email_regex(email): """Basic email validation using a simple regex""" return bool(re.match(r"^[^@]+@^[^@]+\.[^@]+\$", email))</pre>

This task involved completing a Python function using regular expressions. Zero-shot prompting led to a slightly longer function output in Chat-GPT, but Gemini's outputs remained the same across both prompting techniques. However, Gemini included an import line in the zero-shot outline, likely because it was only tasked with returning working code, rather than being asked to iterate through several options and choose the best. Self-consistency led Chat-GPT to condense the function, but the code itself is the same. Both models gave different functions as output - Gemini's isn't as rigid (and will accept far too many incorrectly formatted emails), while Chat-GPT's is much more specific.

Generating Flask APIs (Python)

Zero-shot	Create a '/greet/' endpoint that returns a JSON greeting: [code can be found in RawData.pdf]	<pre> from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet<username>') def greet(username): return jsonify({'message': f'Hello, {username}!'}) # Optional: if you want to run the app directly if __name__ == '__main__': app.run(debug=True) </pre>	<pre> from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet<username>') def greet(username): """Returns a JSON greeting with the provided username.""" greeting_message = f'Hello, {username}! Welcome to Williamsburg, Virginia!' return jsonify({'greeting': greeting_message, 'location': 'Williamsburg, Virginia, United States'}) if __name__ == '__main__': app.run(debug=True) </pre>
Chain-of-thought	Create a '/greet/' endpoint that returns a JSON greeting. Do it step by step : [code can be found in RawData.pdf]	<pre> from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet<username>') def greet(username): return jsonify({'message': f'Hello, {username}!'}) # Optional: run the app if __name__ == '__main__': app.run(debug=True) </pre>	<pre> from flask import Flask, jsonify app = Flask(__name__) @app.route('/greet<username>') def greet(username): greeting_message = f'Hello, {username}! Welcome to Williamsburg, Virginia, United States!' response_data = {'greeting': greeting_message, 'location': 'Williamsburg, Virginia, United States'} return jsonify(response_data) if __name__ == '__main__': app.run(debug=True) </pre>

This task involved generating additional Python code to supplement a given outline. With zero-shot prompting, Gemini included some documentation, and condensed the return function. With chain-of-thought, it did not include documentation, which could be because chain-of-thought requires the model to run through the code step-by-step, which provides the same information as documentation would. Chat-GPT's responses did not change based on the prompting style. Gemini and Chat-GPT gave very similar responses, but Gemini's was tailored more to the specific user.

SQL Schema Design (SQL)

Prompt-chaining	<p>Using the outline, write the schema for a review app with users in psuedocode: -- TODO : Design schema with appropriate keys and constraints -- Tables : users (id , name), books (id , title), reviews (id , user_id , book_id , rating) -></p> <p>Using this psuedocode, implement the schema for a review app with users, books, and reviews: [psuedocode provided by previous response]</p>	<pre> -- USERS TABLE CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Auto-incrementing user ID name TEXT NOT NULL -- User's name); -- BOOKS TABLE CREATE TABLE books (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Auto-incrementing book ID title TEXT NOT NULL -- Book title); -- REVIEWS TABLE CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Auto-incrementing review ID user_id INTEGER NOT NULL, -- References users(id) book_id INTEGER NOT NULL, -- References books(id) rating INTEGER NOT NULL CHECK(rating BETWEEN 1 AND 5), -- Rating 1-5 -- Ensure the same user can't review the same book more than once UNIQUE (user_id, book_id), -- Foreign key constraints FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id)); </pre>	<pre> -- Schema for a Review App -- Table: users CREATE TABLE users (id INTEGER PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255) NOT NULL, email VARCHAR(255) UNIQUE, registration_date DATETIME DEFAULT CURRENT_TIMESTAMP); -- Table: books CREATE TABLE books (id INTEGER PRIMARY KEY AUTO_INCREMENT, title VARCHAR(255) NOT NULL UNIQUE, author VARCHAR(255), publication_year INTEGER, isbn VARCHAR(20) UNIQUE); -- Table: reviews CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTO_INCREMENT, user_id INTEGER NOT NULL, book_id INTEGER NOT NULL, rating INTEGER NOT NULL, review_text TEXT, created_at DATETIME DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), CONSTRAINT rating_range CHECK (rating >= 1 AND rating <= 5), CONSTRAINT unique_user_book UNIQUE (user_id, book_id)); </pre>
Zero-shot	<p>Write the schema for a review app with users, books, and reviews: -- TODO : Design schema with appropriate keys and constraints -- Tables : users (id , name), books (id , title), reviews (id , user_id , book_id , rating)</p>	<pre> -- USERS TABLE CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Unique ID for each user name TEXT NOT NULL -- User's name); -- BOOKS TABLE CREATE TABLE books (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Unique ID for each book title TEXT NOT NULL -- Book title); -- REVIEWS TABLE CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTOINCREMENT, -- Unique ID for each review user_id INTEGER NOT NULL, -- Foreign key to users.id book_id INTEGER NOT NULL, -- Foreign key to books.id rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5), -- Rating: 1 to 5 -- Constraints FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), UNIQUE (user_id, book_id) -- Prevents duplicate reviews per user/book); </pre>	<pre> -- Schema for a Review App -- Table: users CREATE TABLE users (id INTEGER PRIMARY KEY AUTO_INCREMENT, -- Unique identifier for each user name VARCHAR(255) NOT NULL, -- User's name -- Add other user-related fields as needed (e.g., email, registration_date) email VARCHAR(255) UNIQUE, -- Optional: User's email, must be unique if present registration_date DATETIME DEFAULT CURRENT_TIMESTAMP -- Optional: Date when the user registered); -- Table: books CREATE TABLE books (id INTEGER PRIMARY KEY AUTO_INCREMENT, -- Unique identifier for each book title VARCHAR(255) NOT NULL UNIQUE, -- Title of the book, must be unique -- Add other book-related fields as needed (e.g., author, publication_year, ISBN) author VARCHAR(255), -- Optional: Author of the book publication_year INTEGER, -- Optional: Year the book was published isbn VARCHAR(20) UNIQUE -- Optional: ISBN of the book, must be unique if present); -- Table: reviews CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTO_INCREMENT, -- Unique identifier for each review user_id INTEGER NOT NULL, -- Foreign key referencing the users table book_id INTEGER NOT NULL, -- Foreign key referencing the books table rating INTEGER NOT NULL, -- Rating given by the user for the book review_text TEXT, -- Optional: Text content of the review created_at DATETIME DEFAULT CURRENT_TIMESTAMP, -- Date and time when the review was created -- Constraints for data integrity FOREIGN KEY (user_id) REFERENCES users(id), -- Ensures user exists FOREIGN KEY (book_id) REFERENCES books(id), -- Ensures book exists CONSTRAINT rating_range CHECK (rating >= 1 AND rating <= 5), -- Ensures rating is within valid range CONSTRAINT unique_user_book UNIQUE (user_id, book_id) -- Ensures a user can only review a book once); </pre>

This task involved generating an SQL schema using a given outline and parameters. The responses for each model did not vary based on prompting technique, except for the inclusion of comments in Gemini's zero-shot output, and more casual language in Chat-GPT's comments in its zero-prompt output. Gemini gave a more robust schema, with additional data categories, while Chat-GPT's was more bare-bones but followed the specifications that the prompt asked for.

Null Dereference Detection (Java)

Chain-of-thought	<p>Identify any null dereference risk. Do it step by step: [code can be found in RawData.pdf]</p>	<p>Yes, this method contains a null dereference risk. The risk comes from this line: return s.length();</p> <p>It directly calls .length() on s without checking whether s is null.</p>	<p>Based on the analysis, there is a null dereference risk in the <code>getLength</code> function. If the <code>String</code> object passed as the argument <code>s</code> is <code>null</code>, the line <code>return s.length();</code> will throw a <code>NullPointerException</code>.</p> <p>Conclusion: The <code>getLength</code> function has a null dereference risk because it directly calls the <code>length()</code> method on the input <code>String</code> object <code>s</code> without checking if <code>s</code> is <code>null</code>. If <code>s</code> is <code>null</code>, this will lead to a <code>NullPointerException</code>.</p>
------------------	---	--	---

Self-consistency	<p>Use several examples to identify dereference instances in this code, and identify the risk:</p> <p>[code can be found in RawData.pdf]</p>	<p>The line <code>return s.length();</code>, dereferences <code>s</code> and assumes it is not null. This poses a null dereference risk, which occurs if the method is called with a null argument. Let me know if you want to see safe ways to handle this, like default values or exception handling!</p>	<p>Dereference Instance: The dereference instance in the <code>getLength</code> function occurs at the line:</p> <pre>Java return s.length();</pre> <p>Here, the code attempts to access a member (<code>length()</code>) of the object referenced by the variable <code>s</code>.</p> <p>Risk: The primary risk associated with this dereference instance is a NullPointerException. This is a runtime exception that occurs when you try to access a member (field or method) of an object through a null reference.</p> <p>Consequences of NullPointerException:</p> <ul style="list-style-type: none">• Program Termination: If a NullPointerException is not caught and handled, it will cause the program to abruptly terminate.• Loss of Data: If the program was in the middle of an important operation, data might be lost.• Poor User Experience: For end-users, the application will crash, leading to a frustrating experience.• Debugging Challenges: While the location of a NullPointerException is usually clear from the stack trace, identifying the exact reason why a variable became null can sometimes require careful debugging. <p>In summary, the <code>getLength</code> function, as written, is susceptible to a NullPointerException if a null String reference is passed as an argument. This highlights the importance of null checks when dealing with object references in programming.</p>
------------------	--	--	---

Zero-shot	<p>Improve the parser to support quoted fields:</p> <p>[code can be found in RawData.pdf]</p>	<pre>import csv from io import StringIO def parse_csv_line(line): reader = csv.reader(StringIO(line)) return next(reader)</pre>	<pre>import re def parse_csv_line_improved(line): """ Parses a CSV line, correctly handling quoted fields. Args: line: A string representing a single line from a CSV file. Returns: A list of strings, where each string is a field from the CSV line. """ # Regex to find fields, handling quoted values # Explanation: # (?:"(?:\\. [^"])*" (?:\\. [^,]*)?) - Non-capturing group # (?:\\. [^,]*)? - Match a quoted field (content inside double quotes) - Group 1 # (?:\\. [^,]*)? - Match a non-quoted field (any character except comma) - Group 2 #) - End of non-capturing group # (?!,) - Match a comma OR the end of the line (non-capturing group) pattern = r'(?:"(?:\\. [^"])*" (?:\\. [^,]*)?) (?:\\. [^,]*)?(?!,)' fields = re.findall(pattern, line) # Process the result to get the actual field values return [field[0] if field[0] else field[1] for field in fields]</pre>
Chain-of-thought	<p>Improve the parser to support quoted fields. Do it step-by-step:</p> <p>[code can be found in RawData.pdf]</p>	<pre>def parse_csv_line(line): fields = [] field = "" in_quotes = False i = 0 while i < len(line): char = line[i] if char == '"': if in_quotes and i + 1 < len(line) and line[i + 1] == '"': # Escaped quote: "" inside quoted field field += '"' i += 1 else: in_quotes = not in_quotes # Toggle quote mode elif char == ',' and not in_quotes: fields.append(field) field = "" else: field += char i += 1 fields.append(field) # Add the last field return fields</pre>	<pre>import csv def parse_csv_line_using_csv(line): reader = csv.reader([line]) # Create a reader for a single line for row in reader: return list(row) # Return the row as a list return [] # Handle empty lines</pre>

<p>Zero-shot</p>	<p>Convert the data class to a REST API using Ktor:</p> <p>[code can be found in RawData.pdf]</p>	<pre>import io.ktor.application.* import io.ktor.features.* import io.ktor.http.* import io.ktor.request.* import io.ktor.response.* import io.ktor.routing.* import io.ktor.serialization.* import kotlinx.serialization.* import kotlinx.serialization.json.* @kotlinx.serialization.Serializable data class Product(val id: Int, val name: String, val price: Double) val products = mutableListOf<Product>() fun Application.module() { install(ContentNegotiation) { json() } routing { route("/products") { // GET /products get { call.respond(products) } // POST /products post { val product = call.receive<Product>() products.add(product) call.respond(HttpStatusCode.Created, product) } } } }</pre>	<pre>import io.ktor.server.application.* import io.ktor.server.response.* import io.ktor.server.routing.* import io.ktor.server.netty.* import io.ktor.serialization.kotlinx.json.* import io.ktor.server.plugins.contentnegotiation.* import io.ktor.server.request.* import kotlinx.serialization.Serializable import io.ktor.http.HttpStatusCode @Serializable data class Product(val id: Int, val name: String, val price: Double) // In-memory storage for products (for demonstration purposes) val products = mutableListOf<Product>() { Product(1, "Laptop", 1200.0), Product(2, "Mouse", 25.0), Product(3, "Keyboard", 75.0) } fun main(args: Array<String>) { io.ktor.server.netty.EngineMain.main(args) } fun Application.module() { // Install the ContentNegotiation plugin to handle JSON serialization install(ContentNegotiation) { json() } routing { // GET /products: Returns a list of all products get("/products") { call.respond(products) } // GET /products/{id}: Returns a single product by ID get("/products/{id}") { val id = call.parameters["id"]?.toIntOrNull() if (id == null) { call.respond(HttpStatusCode.BadRequest, "Product ID must be an integer.") } } } }</pre>
------------------	---	---	---

			<pre> return@get } val product = products.find { it.id == id } if (product != null) { call.respond(product) } else { call.respond(HttpStatusCode.NotFound, "Product not found.") } } // POST /products: Creates a new product post("/products") { val product = try { call.receive<Product>() } catch (e: Exception) { call.respond(HttpStatusCode.BadRequest, "Invalid product data format. Expected JSON: { id: Int, name: String, price: Double }") return@post } //Basic validation if (product.id <= 0 product.name.isBlank() product.price <= 0) { call.respond(HttpStatusCode.BadRequest, "Invalid product data. id and price must be positive, name cannot be blank.") return@post } // Check if product ID already exists if (products.any { it.id == product.id }) { call.respond(HttpStatusCode.Conflict, "Product with this ID already exists.") return@post } products.add(product) call.respond(HttpStatusCode.Created, product) // Respond with 201 Created and the new product } </pre>
Chain-of-thought	<p>Convert the data class to a REST API using Ktor. Do it step by step:</p> <p>[code can be found in RawData.pdf]</p>	<pre> import io.ktor.application.* import io.ktor.features.ContentNegotiation import io.ktor.http.HttpStatusCode import io.ktor.request.receive import io.ktor.response.respond import io.ktor.routing.* import io.ktor.server.engine.embeddedServer import io.ktor.server.netty.Netty import io.ktor.server.plugins.contentnegotiation.ContentNegotiation import io.ktor.server.plugins.statuspages.StatusPages import io.ktor.server.testing.testApplication import io.ktor.serialization.json @kotlinx.serialization.Serializable data class Product(val id: Int, val name: String, val price: Double) val products = mutableListOf<Product>() // In-memory list to store products fun Application.module() { install(ContentNegotiation) { json() // Set up JSON serialization } routing { route("/products") { // GET endpoint to fetch all products get { if (products.isEmpty()) { call.respond(HttpStatusCode.NoContent, "No products found") } else { call.respond(products) } } // POST endpoint to add a new product post { val product = call.receive<Product>() products.add(product) call.respond(HttpStatusCode.Created, product) } } } } </pre>	<pre> import io.ktor.server.application.* import io.ktor.server.response.* import io.ktor.server.routing.* import io.ktor.server.netty.* // Use Netty import io.ktor.serialization.kotlinx.json.* // Use kotlinx-json import io.ktor.server.plugins.contentnegotiation.* // For ContentNegotiation import io.ktor.server.request.* // For receiving request bodies import kotlinx.serialization.Serializable // For JSON serialization import io.ktor.http.HttpStatusCode // For HTTP status codes // 1. Define the Data Class and Mark it Serializable @Serializable data class Product(val id: Int, val name: String, val price: Double) // 2. In-Memory Data Storage (Replace with a database in a real app) val products = mutableListOf<Product>{ Product(1, "Laptop", 1200.0), Product(2, "Mouse", 25.0), Product(3, "Keyboard", 75.0) } // 3. Application Entry Point (main function) fun main(args: Array<String>) { io.ktor.server.netty.EngineMain.main(args) // Start Netty } // 4. Application Module (Configure Ktor) fun Application.module() { // 4.1 Install Content Negotiation (for JSON) install(ContentNegotiation) { json() // Use kotlinx-json for JSON serialization/deserialization } // 4.2 Define Routes (Endpoints) routing { // 4.2.1 GET /products (Get All Products) get("/products") { call.respond(products) // Respond with the product list (as JSON) } // 4.2.2 GET /products/{id} (Get Product by ID) get("/products/{id}") { val id = call.parameters["id"]?.toIntOrNull() // Get ID from path if (id == null) { call.respond(HttpStatusCode.BadRequest, "Invalid product ID") return@get // Exit if ID is not an integer } val product = products.find { it.id == id } // Find product by ID if (product != null) { call.respond(product) // Respond with product (as JSON) } else { call.respond(HttpStatusCode.NotFound, "Product not found") } } // 4.2.3 POST /products (Create a New Product) post("/products") { val product = call.receive<Product>() // Get Product from JSON in request body // Validation: Check for valid data if (product.id <= 0 product.name.isBlank() product.price <= 0) { call.respond(HttpStatusCode.BadRequest, "Invalid product data. ID and price must be positive, name cannot be blank.") return@post } // Check for duplicate IDs if (products.any { it.id == product.id }) { call.respond(HttpStatusCode.Conflict, "Product with this ID already exists.") return@post } products.add(product) // Add the new product call.respond(HttpStatusCode.Created, product) // Respond with 201 Created } } } </pre>

This task involved converting a data class into an API using a specific framework, KTOR. The responses across both prompting methods are nearly identical. Both of Gemini's responses are more detailed, robust, and include examples built in. Chat-GPT's are more bare-bones but work similarly.

Function Summarization (Python)

Zero-shot	Write a brief summary of the function:	The function <code>reverse_words(sentence)</code> takes a string <code>sentence</code> as input and returns a new string where	The <code>reverse_words</code> function takes a string (<code>sentence</code>) as input, splits it into individual words,
-----------	--	--	---

	[code can be found in RawData.pdf]	the words in the sentence are reversed in order. It does so by splitting the sentence into words, reversing the order of the words, and then joining them back together with spaces.	reverses the order of those words, and then joins the reversed words back into a single string, which it returns. Essentially, it reverses the order of words in a sentence.
Few-shot	<p>Example 1:</p> <pre>def is_even(n): if n % 2 == 0: return True else: return False This method determines whether a number is even. <p>Example 2:</p> <pre>def is_prime(n): if n <= 1: return False for i in range(2, n): if n%i == 0: return False return True This method determines if a number is prime. <p>Write a brief summary of the function:</p> <p>[code can be found in RawData.pdf]</p> </pre></pre>	The function <code>reverse_words(sentence)</code> takes a string <code>sentence</code> and returns a new string with the words reversed in order. It splits the sentence into words, reverses the list of words, and then joins them back together with spaces.	The <code>reverse_words</code> function takes a sentence (string) as input and returns a new string with the order of the words reversed.

This task involved summarizing a short Python function. Chat-GPT's responses are very similar across both prompting methods. Gemini's output for zero-shot is slightly lengthier and more dense than its output for few-shot prompting. The language used in the examples in few-shot is more casual, and only contains one simple sentence, so it formulated its response to be similar. The responses across both models are very similar.

Prompt from Code Comments (Python)

Zero-shot	<p>Write a prompt that could generate the code:</p> <p>[code can be found in RawData.pdf]</p>	<p>Prompt: "Write a Python function named <code>is_prime</code> that takes a number <code>n</code> as input and returns <code>True</code> if the number is prime, and <code>False</code> otherwise. The function should check if the number is less than or equal to 1 and return <code>False</code> in that case. For numbers greater than 1, the function should check divisibility only up to the square root of <code>n</code> to optimize the process."</p>	"Write a Python function called <code>is_prime</code> that takes an integer <code>n</code> as input and returns <code>True</code> if <code>n</code> is a prime number, and <code>False</code> otherwise. Optimize the primality check by only iterating up to the square root of <code>n</code> ."
Few-shot	<p>Example 1:</p> <p>Code:</p> <pre># This function checks if a number is even. def is_even(n): if n % 2 == 0: return True else: return False Prompt: Write a function that checks if a number is even. <p>Example 2:</p> <p>Code:</p> <pre># This function checks if a number is odd. def is_odd(n): if n % 2 == 0: return False else: return True Prompt: Write a function that checks if a number is odd. <p>Write a prompt that could generate the code:</p> <p>[code can be found in RawData.pdf]</p> </pre></pre>	<p>Prompt: "Write a function that checks if a number is prime. The function should return <code>True</code> if the number is prime and <code>False</code> otherwise. It should handle edge cases like numbers less than or equal to 1 and efficiently check divisibility by iterating from 2 up to the square root of the number."</p>	"Write a Python function called <code>is_prime</code> that takes an integer <code>n</code> as input and returns <code>True</code> if <code>n</code> is a prime number, and <code>False</code> otherwise. Optimize the primality check by only iterating up to the square root of <code>n</code> ."

This task involved reverse-engineering a prompt from given Python code. Chat-GPT's output for zero-shot is much more detailed, but is comparatively similar to its response for few-shot. Gemini's responses were identical across both prompt methods. Gemini's responses are more technical and to-the-point than Chat-GPT's, which are more conversational.

Fixing Factorial Bug (Python)

Chain-of-thought	<p>Fix the bug when input is 0. Do it step by step:</p> <p>[code can be found in RawData.pdf]</p>	<pre>def factorial(n): if n == 0: return 1 # Factorial of 0 is 1 result = 1 for i in range(1, n + 1): # Loop from 1 to n, inclusive result *= i return result</pre>	<pre>def factorial(n): """ Calculates the factorial of a non-negative integer. Args: n: A non-negative integer. Returns: The factorial of n, or 1 if n is 0. Raises ValueError if n is negative. """ if n < 0: raise ValueError("Factorial is not defined for negative numbers") if n == 0: return 1 result = 1 for i in range(1, n + 1): result *= i return result</pre>
Self-consistency	<p>Testing several different fixes, and finding the most efficient one: fix the bug when input is 0.</p> <p>[code can be found in RawData.pdf]</p>	<pre>import math def factorial(n): return math.factorial(n)</pre>	<pre>import math def factorial_v3(n): """Calculates the factorial of a non-negative integer using math.factorial.""" if n < 0: raise ValueError("Factorial is not defined for negative numbers") return math.factorial(n)</pre>

This task involved fixing a bug in a Python function. With chain-of-thought prompting, the responses are lengthier and contain more useful documentation. This is interesting because so far, chain-of-thought has actually led to less documentation in the other tasks. Both outputs for self-consistency prompting led to the use of `math.factorial`, which makes sense because the prompt asks the models to find the most efficient method. Both Gemini's and Chat-GPT's responses for chain-of-thought and self-consistency are the same, but Gemini added an additional line to check for a value error, making it a better response.

Linked List Node Deletion (C)

Zero-shot	<p>Implement node deletion by value:</p> <p>[code can be found in RawData.pdf]</p>	<pre>#include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; // Function to delete the node by value (key) void deleteNode(struct Node** head, int key) { // Step 1: Check if the list is empty if (*head == NULL) { printf("List is empty.\n");</pre>	<pre>#include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { struct Node *current = *head; struct Node *prev = NULL; // Handle the case where the list is empty</pre>
-----------	--	---	--

		<pre> return; } // Step 2: If the node to be deleted is the head node if ("head->data == key) { struct Node* temp = "head; "head = ("head->next; // Update the head to the next node free(temp); // Free the memory of the deleted node return; } // Step 3: Traverse the list to find the node to be deleted struct Node* current = "head; while (current->next != NULL && current->next->data != key) { current = current->next; // Move to the next node } // Step 4: If the node with the given key was found if (current->next != NULL) { struct Node* temp = current->next; current->next = current->next->next; // Skip the node to be deleted free(temp); // Free the memory of the deleted node } else { printf("Node with value %d not found.\n", key); // If the key is not found } } </pre>	<pre> if ("head == NULL) { return; // Nothing to delete } // Handle the case where the key is in the head node if (current != NULL && current->data == key) { "head = current->next; // Change head to the next node free(current); // Free the old head node return; } // Search for the key in the rest of the list while (current != NULL && current->data != key) { prev = current; current = current->next; } // If the key was not found if (current == NULL) { return; // Key not found in the list } // Unlink the node from the list prev->next = current->next; // Free the memory of the node to be deleted free(current); } </pre>
Chain-of-thought	Implement node deletion by value. Do it step-by-step . [code can be found in RawData.pdf]	<pre> #include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node** head, int key) { // Step 1: Check if the list is empty if ("head == NULL) { printf("List is empty.\n"); return; } // Step 2: If the node to be deleted is the head node if ("head->data == key) { struct Node* temp = "head; "head = ("head->next; // Move the head to the next node free(temp); // Free the memory of the deleted node return; } // Step 3: Traverse the list to find the node to be deleted struct Node* current = "head; while (current->next != NULL && current->next->data != key) { current = current->next; // Move to the next node } // Step 4: If the node with the given key was found if (current->next != NULL) { struct Node* temp = current->next; current->next = current->next->next; // Skip the node to be deleted free(temp); // Free the memory of the deleted node } else { printf("Node with value %d not found.\n", key); // If the key is not found } } </pre>	<pre> #include <stdio.h> #include <stdlib.h> struct Node { int data; struct Node* next; }; void deleteNode(struct Node **head, int key) { // Step 1: Handle the empty list case. If the list is empty, there's nothing to do. if ("head == NULL) { return; } struct Node *current = "head; // Start at the head struct Node *prev = NULL; // Keep track of the previous node // Step 2: Handle the case where the node to delete is the head node. if (current != NULL && current->data == key) { "head = current->next; // Change the head pointer to point to the second node. free(current); // Free the memory of the original head node. return; // We're done, so return. } // Step 3: Iterate through the list to find the node to delete. // We need to keep track of the previous node (prev) so we can // unlink the node to be deleted. while (current != NULL && current->data != key) { prev = current; // Move 'prev' to the current node current = current->next; // Move 'current' to the next node } // Step 4: Handle the case where the key was not found. if (current == NULL) { return; // The key was not found in the list. } // Step 5: Unlink the node from the list and free the memory. // 'current' now points to the node to be deleted, and 'prev' // points to the node before it. prev->next = current->next; // Make the previous node point to the node after 'current' free(current); // Free the memory occupied by the node to be deleted. } </pre>

This task involved implementing an additional function to pre-written C code. Chat-GPT's responses for both prompt methods were the exact same. Gemini's response for zero-shot has a small logic error when handling the deletion of a node that isn't the head, but the issue is not present in it's response for chain-of-thought. Gemini's responses contain more documentation than Chat-GPT's, which is a common theme amongst these tasks so far.

Recursive Function Completion (Python)

Zero-shot	Complete the recursive function for Fibonacci. [code can be found in RawData.pdf]	<pre> def fibonacci(n): # Base cases if n == 0: return 0 elif n == 1: return 1 # Recursive case return fibonacci(n - 1) + fibonacci(n - 2) </pre>	<pre> def fibonacci(n): """ Calculates the nth Fibonacci number using recursion. Args: n: A non-negative integer. Returns: The nth Fibonacci number. Raises: ValueError: If n is negative. """ if n < 0: raise ValueError("Fibonacci is not defined for negative numbers") if n == 0: return 0 elif n == 1: return 1 else: return fibonacci(n - 1) + fibonacci(n - 2) </pre>
Chain-of-thought	Complete the recursive function for Fibonacci. Do it step by step . [code can be found in RawData.pdf]	<pre> def fibonacci(n): # Step 1: Base cases if n == 0: return 0 elif n == 1: return 1 # Step 2: Recursive case return fibonacci(n - 1) + fibonacci(n - 2) </pre>	<pre> def fibonacci(n): """ Calculates the nth Fibonacci number using recursion. Args: n: A non-negative integer. Returns: The nth Fibonacci number. Raises: ValueError: If n is negative. """ # Step 1: Handle the error case: negative input if n < 0: raise ValueError("Fibonacci is not defined for negative numbers") # Step 2: Handle the base cases: F(0) = 0, F(1) = 1 if n == 0: return 0 # Base case: 0th Fibonacci number is 0 elif n == 1: return 1 # Base case: 1st Fibonacci number is 1 </pre>

			<pre># Step 3: Recursive step: F(n) = F(n-1) + F(n-2) # else: return fibonacci(n - 1) + fibonacci(n - 2) # Recursive call</pre>
--	--	--	---

This task involved completing a Python function. Chat-GPT's responses are identical for chain-of-thought and zero-shot prompting, but chain-of-thought includes comments detailing the exact step the code is on. The same goes for Gemini's responses. Between Chat-GPT and Gemini, there are no significant differences in the responses other than Gemini containing more detailed documentation.

Constructor Completion (Python)

Zero-shot	Complete the class constructor. [code can be found in RawData.pdf]	<pre>class Person: def __init__(self, name, age, email=None): self.name = name # Required: name of the person self.age = age # Required: age of the person self.email = email # Optional: email of the person (default is None)</pre>	<pre>class Person: """ Represents a person with a name, age, and optional email address. """ def __init__(self, name, age, email=None): """ Constructs a Person object. Args: name: The person's name (string). age: The person's age (integer). email: The person's email address (string, optional). """ # Input Validation: Check that name is a string and age is an integer. if not isinstance(name, str): raise TypeError("Name must be a string") if not isinstance(age, int): raise TypeError("Age must be an integer") if age < 0: raise ValueError("Age cannot be negative") if email is not None and not isinstance(email, str): raise TypeError("Email must be a string or None") self.name = name self.age = age self.email = email # Can be None</pre>
Chain-of-thought	Complete the class constructor. Do it step by step . [code can be found in RawData.pdf]	<pre>class Person: def __init__(self, name, age, email=None): self.name = name # Required: name of the person self.age = age # Required: age of the person self.email = email # Optional: email of the person (default is None)</pre>	<pre>class Person: """ Represents a person with a name, age, and optional email address. """ def __init__(self, name, age, email=None): """ Constructs a Person object. Args: name: The person's name (string). age: The person's age (integer). email: The person's email address (string, optional). """ # Step 1: Input Validation # Check if the provided name is a string if not isinstance(name, str): raise TypeError("Name must be a string") # Check if the provided age is an integer and is not negative if not isinstance(age, int) or age < 0: raise ValueError("Age must be a non-negative integer") # Check if email is provided and is a string if email is not None and not isinstance(email, str): raise TypeError("Email must be a string or None") # Step 2: Assign attributes to the Person object # Assign the name to the object's name attribute self.name = name # Assign the age to the object's age attribute self.age = age # Assign the email to the object's email attribute # It will be None if no email was provided. self.email = email</pre>

This task involved completing a Python constructor given some parameters. Both Gemini's and Chat-GPT's responses are identical across both prompting methods, but Gemini's response for chain-of-thought prompting includes comments regarding the step the code is on. Compared to each other, Gemini's contains more documentation and handles some possible errors.

Binary Search Completion (Java)

Zero-shot	Complete the binary search implementation. [code can be found in RawData.pdf]	<pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; // Compare and adjust bounds if (arr[mid] == target) { return mid; // Found the target at index mid } else if (arr[mid] < target) { left = mid + 1; // Target is in the right half } else { right = mid - 1; // Target is in the left half } } return -1; // Target not found }</pre>	<pre>public class BinarySearch { /** * Performs a binary search on a sorted integer array. * * @param arr The sorted integer array to search. * @param target The integer value to search for. * @return The index of the target value in the array if found, otherwise -1. */ public static int binarySearch(int[] arr, int target) { int left = 0; int right = arr.length - 1; while (left <= right) { int mid = left + (right - left) / 2; // Prevent potential overflow // int mid = (left + right) / 2; //Original // TODO: Compare and adjust bounds if (arr[mid] == target) { return mid; // Target found at mid index } else if (arr[mid] < target) { left = mid + 1; // Target is in the right half } else { right = mid - 1; // Target is in the left half } } return -1; // Target not found } }</pre>
Chain-of-thought	Complete the binary search implementation. Do it step by step . [code can be found in RawData.pdf]	<pre>public int binarySearch(int[] arr, int target) { int left = 0, right = arr.length - 1; while (left <= right) { int mid = (left + right) / 2; // Calculate the midpoint // Compare the target with arr[mid] if (arr[mid] == target) { return mid; // Target found, return the index } else if (arr[mid] < target) { left = mid + 1; // Target is in the right half, move left bound } else { right = mid - 1; // Target is in the left half, move right bound } } }</pre>	<pre>public class BinarySearch { /** * Performs a binary search on a sorted integer array. * * @param arr The sorted integer array to search. * @param target The integer value to search for. * @return The index of the target value in the array if found, otherwise -1. */ public static int binarySearch(int[] arr, int target) { // Step 1: Initialize left and right pointers. int left = 0; // Start at the beginning of the array. int right = arr.length - 1; // End at the last element of the array. // Step 2: Iterate while the search space is valid (left <= right). while (left <= right) {</pre>

		<pre>return -1; // Target not found }</pre>	<pre>// Step 3: Calculate the middle index. This improved version prevents overflow. int mid = (left + (right - left)) / 2; // int mid = (left + right) / 2; // Original way, can cause overflow // Step 4: Compare the middle element with the target. if (arr[mid] == target) { return mid; // Step 4a: Target found! Return the index. } else if (arr[mid] < target) { // Step 4b: If the middle element is less than the target, // the target must be in the right half of the array. left = mid + 1; // Update left to search the right half. } else { // Step 4c: If the middle element is greater than the target, // the target must be in the left half of the array. right = mid - 1; // Update right to search the left half. } // Step 5: Target not found. If the loop finishes without finding the target, // it's not in the array. return -1; }</pre>
--	--	---	---

This task involved completing a binary search function, specifically finetuning the comparing and adjusting of bounds. The responses were almost identical for both prompting methods, but Gemini's output for chain-of-thought prompting has much more detailed documentation and comments throughout the function. Gemini's output was functionally similar but contained within a class, whereas Chat-GPT's output was just the function alone.

Self-Consistency Bug Fixing (C++)

Zero-shot	Resolve inconsistency between function name and logic: [code can be found in RawData.pdf]	<pre>// Correct function name and logic to return true if x is odd bool isOdd(int x) { return x % 2 != 0; // Return true if x is odd }</pre>	<pre>// Solution 1: Rename the function // Returns true if x is even bool isEven(int x) { return x % 2 == 0; } // Solution 2: Correct the logic // Returns true if x is odd bool isOdd(int x) { return x % 2 != 0; }</pre>
Self-consistency	Provide several input and output examples, and briefly summarize the function based on those examples. Then, determine whether there is an inconsistency between function name and logic. Resolve the inconsistency if present: [code can be found in RawData.pdf]	<p>Based on the examples, the function <code>isOdd</code> is currently behaving like a function that checks if a number is even, not odd. However, the function name <code>isOdd</code> contradicts this behavior, as it implies the function should return <code>true</code> for odd numbers. There is indeed an inconsistency between the function name <code>isOdd</code> and its logic.</p> <ul style="list-style-type: none">Name <code>isOdd</code> implies the function should return <code>true</code> for odd numbers.Logic (<code>x % 2 == 0</code>) checks if a number is even, so it returns <code>true</code> for even numbers. <p>Option 1: Change the Function Name to <code>isEven</code> // Correct name and logic for even check bool isEven(int x) { return x % 2 == 0; // Return true if x is even }</p> <p>Option 2: Change the Logic to Match <code>isOdd</code> // Keep the function name <code>isOdd</code>, but update the logic to check for odd numbers bool isOdd(int x) { return x % 2 != 0; // Return true if x is odd }</p>	<pre>// Supposed to return true if x is even bool isOdd(int x) { return x % 2 == 0; // Logic contradicts function name } #include <iostream> int main() { // The corrected function bool isOddCorrected(int x) { return x % 2 != 0; } // Alternative Resolution (Rename the function) bool isEven(int x) { return x % 2 == 0; } }</pre>

This task involved fixing an inconsistency in a given C++ function. It is a useful measure because the inconsistency has two different fixes, and the outputs given can show how the model addresses this. Zero-shot prompting led to shorter responses compared to self-consistency. Chat-GPT's response for zero-shot prompting includes only one of the two potential fixes, whereas Gemini's responses for both prompting methods include both fixes. Gemini's outputs include more documentation within the code itself, while Chat-GPT's have more essay-based explanations.

Prompt Chaining: Bug Identification → Fix (JavaScript)

Prompt-chaining	Identify the bug in this code (do not fix it): [code can be found in RawData.pdf] -> Fix the [previous response] bug in this code: [code can be found in RawData.pdf]	<p>Option 1: Change the function name to <code>isEven</code> (since the logic checks for even numbers). // Corrected: Function name matches its logic to check for even numbers bool isEven(int x) { return x % 2 == 0; // Return true if x is even }</p> <p>Option 2: Change the logic to match the function name <code>isOdd</code> (if the intent is to check for odd numbers). // Corrected: Function name matches its logic to check for odd numbers bool isOdd(int x) { return x % 2 != 0; // Return true if x is odd }</p>	<pre>// Solution 1: Rename the function to isEven bool isEven(int x) { return x % 2 == 0; } // Solution 2: Correct the logic to check for odd numbers bool isOdd(int x) { return x % 2 != 0; }</pre>
Chain-of-thought	Identify and fix the bug. Do it step by step : [code can be found in RawData.pdf]	<p>The function name <code>isOdd</code> suggests that it is supposed to return <code>true</code> if <code>x</code> is odd. However, the logic <code>x % 2 == 0</code> checks if <code>x</code> is even, returning <code>true</code> for even numbers. This creates a contradiction between the function name and its logic.</p> <p>There are two ways to resolve this issue: Option 1: Change the Function Name to <code>isEven</code> If the intent of the function is to check whether <code>x</code> is even, the function name should be changed to <code>isEven</code>. The logic of checking for even numbers (<code>x % 2 == 0</code>) is correct. // Corrected: Change function name to <code>isEven</code> to match the logic bool isEven(int x) { return x % 2 == 0; // Return true if x is even }</p> <p>Option 2: Change the Logic to Match <code>isOdd</code> If the function is supposed to check whether <code>x</code> is odd, the logic should be updated to check if the number is odd, which is done by using <code>x % 2 != 0</code>. // Corrected: Change the logic to check if <code>x</code> is odd bool isOdd(int x) { return x % 2 != 0; // Return true if x is odd }</p>	<p>The code is intended to return <code>true</code> if <code>x</code> is even, but the function is named <code>isOdd</code>. The logic within the function checks for evenness (<code>x % 2 == 0</code>) which contradicts the function name.</p> <p>If the intended behavior is to check for even numbers, we should rename the function:</p> <pre>bool isEven(int x) { return x % 2 == 0; }</pre> <p>If the intended behavior is to check for odd numbers, we should correct the logic:</p> <pre>bool isOdd(int x) { return x % 2 != 0; }</pre>

This task involves identifying and fixing a bug in a JavaScript function. Similar to the previous task, there is an inconsistency that can be resolved in two main ways. This prompt addresses it as a “bug”, which would imply that the best fix is to change the implementation rather than the name of the function, and it would be interesting to see how the outputs vary compared to the previous task. Chat-GPT's responses are similar, but chain-of-thought contains more explanation. Gemini's outputs have the same characteristic. Both models responded in very similar ways, with only mild differences in formatting.

Summary Decomposition (C++)

Zero-shot	Decompose the high-level comment/summary into logical steps: [code can be found in RawData.pdf]	<p>High-Level Comment:</p> <ul style="list-style-type: none">"Function that validates an input, calculates square, and returns result" <p>Step-by-Step Decomposition:</p> <ol style="list-style-type: none">Validate the input:<ul style="list-style-type: none">The function checks if the input <code>x</code> is valid. In this case, the validation is done by checking if <code>x</code> is negative.If <code>x</code> is negative (<code>x < 0</code>), the function returns <code>-1</code> to indicate an invalid input.	<p>Here's a breakdown of the logical steps the process function performs:</p> <ol style="list-style-type: none">Check for negative input: It checks if the input integer <code>x</code> is less than 0.Handle negative input: If <code>x</code> is negative, the function immediately returns <code>-1</code>.Calculate the square: If <code>x</code> is not
-----------	--	--	--

		<div><div><div>2. Calculate the square:</div><div>○ If the input x is valid (i.e., not negative), the function proceeds to calculate the square of x. The square of x is calculated by multiplying x by itself ($x * x$).</div></div><div><div>3. Return the result:</div><div>○ The function returns the result of the square of x if the input was valid. Otherwise, it returns -1 for invalid input.</div></div><div>Logical Steps in Code:<div><div>1. Check if x is negative:</div><div>○ If x is less than 0, return -1.</div></div><div><div>2. Otherwise, calculate the square of x:</div><div>○ If x is valid (non-negative), compute $x * x$ and return the result.</div></div></div></div>	<div>4. negative (i.e., it's zero or positive), the function calculates the square of $x (x * x)$. Return the result: The function returns the calculated square (if x was not negative) or -1 (if x was negative).</div>
Few-shot	<div>Example 1: Code: // Function that returns true if a number is even bool isEven(int number) { return number % 2 == 0; }</div> <div>Steps: Receives a numerical input, divides by 2, returns true if the remainder is 0 (indicating that the number is even), or returns false if the remainder is not 0 (indicating that the number is not even)</div> <div>Example 2: Code: // Function that returns true if a number is odd bool isOdd(int number) { return number % 2 != 0; }</div> <div>Steps: Receives a numerical input, divides by 2, returns true if the remainder is not 0 (indicating that the number is odd), or returns false if the remainder is 0 (indicating that the number is not odd)</div> <div>Decompose the high-level comment/summary into logical steps: [code can be found in RawData.pdf]</div>	<div>High-Level Comment: "Function that validates an input, calculates square, and returns result" Step-by-Step Decomposition:<div><div>1. Validate the input:</div><div>○ The function first checks if the input value x is valid.</div><div>○ In this case, it validates whether x is negative.</div><div>○ If x is negative, the function returns -1 (indicating invalid input).</div></div><div><div>2. Calculate the square:</div><div>○ If the input is valid (i.e., x is not negative), the function proceeds to calculate the square of x.</div><div>○ This is done by multiplying x by itself ($x * x$).</div></div><div><div>3. Return the result:</div><div>○ The function returns the result of $x * x$ (the square) if x was valid.</div><div>○ If x was negative, it returns -1 as an indication of invalid input.</div></div></div>	<div>Steps: Receives a numerical input x, checks if x is less than 0. If x is less than 0, the function returns -1. Otherwise, the function calculates the square of $x (x * x)$ and returns the result.</div>

This task involves summarizing a C++ function into steps, using a given summary as a comment. Chat-GPT's responses are very similar for both zero- and few-shot prompting, with the only significant difference being that the zero-shot output contains additional logical steps in the code to supplement the decomposition. Gemini's responses vary much more across the two prompting styles - zero-shot led to a more thorough answer, with a numbered list of steps and a full breakdown, while few-shot led to a short, concise summary of the steps. Across the two models, Chat-GPT has a more detailed answer and remained more consistent across both prompting methods. Gemini's responses likely differed across the prompting methods because the examples in few-shot are very short and concise, guiding the model to do the same.

Purpose Inference → Completion (Python)

Zero-shot	<div>Complete the function based on intent: [code can be found in RawData.pdf]</div>	<div>def calculate_average(scores): total = sum(scores) # Sum all the scores average = total / len(scores) # Divide the total by the number of scores return average</div>	<div>def calculate_average(scores): """ Calculates the average of a list of numerical scores. Args: scores: A list of numbers (integers or floats). Returns: The average of the scores (float), or 0 if the list is empty. Raises: TypeError: If the input is not a list. ValueError: If the list contains non-numerical values. """ if not isinstance(scores, list): raise TypeError("Input must be a list.") if not scores: # Check for empty list return 0.0 # Return 0.0 for an empty list, as specified total = 0 for score in scores: if not isinstance(score, (int, float)): raise ValueError("All elements in the list must be numbers.") total += score return float(total) / len(scores)</div>
Chain-of-thought	<div>Complete the function based on intent. Do it step by step: [code can be found in RawData.pdf]</div>	<div>def calculate_average(scores): # Step 1: Sum the scores total = sum(scores) # Calculates the sum of the scores # Step 2: Check if the list is empty to avoid division by zero if len(scores) == 0: return 0 # Return 0 or handle it as needed # Step 3: Calculate the average average = total / len(scores) # Step 4: Return the average return average</div>	<div>def calculate_average(scores): """ Calculates the average of a list of numerical scores. Args: scores: A list of numbers (integers or floats). Returns: The average of the scores (float), or 0.0 if the list is empty. Raises: TypeError: If the input is not a list. ValueError: If the list contains non-numerical values. """ # Step 1: Input Validation: Check if the input is a list if not isinstance(scores, list): raise TypeError("Input must be a list.") # Step 2: Handle the empty list case if not scores: return 0.0 # Return 0.0 for an empty list # Step 3: Initialize the total sum of the scores total = 0 # Step 4: Iterate through the scores and accumulate the sum, validating each score for score in scores: if not isinstance(score, (int, float)): raise ValueError("All elements in the list must be numbers.") total += score # Step 5: Calculate the average average = float(total) / len(scores) # Ensure float division # Step 6: Return the calculated average return average</div>

This task involves completing a Python function based on "intent". Chat-GPT's output for the chain-of-thought prompt handles an input of an empty list, whereas the response for the zero-shot prompt does not. Forcing the model to work through the various steps likely led to this difference in response. Gemini's responses are essentially identical across both prompting methods, with the addition of commented steps in the chain-of-thought output. Gemini's outputs handle errors better, and contain more comprehensive documentation, whereas Chat-GPT only handles empty-list errors, and only in the chain-of-thought prompt output.

BONUS Full-File Bug Detection and Refactoring (Python)

Chain-of-thought	<p>Analyze the following utility script. Identify any potential logic issues or design flaws. Then, refactor it for better readability, correctness, and safety. Do it step by step.</p> <p>[code can be found in RawData.pdf]</p>	<p>Analysis of the Utility Script</p> <ol style="list-style-type: none">read_csv() function:<ul style="list-style-type: none">File Handling: The <code>open()</code> function does not have any error handling, so if the file does not exist, it will raise an exception, causing the program to crash.CSV Parsing: The function assumes that the CSV is well-formed. If the file is not properly formatted or contains non-UTF-8 characters, the script will raise an error.summarize_column() function:<ul style="list-style-type: none">Skipping Header: The code uses slicing (<code>data[1:]</code>) to skip the header row. This works, but there are edge cases where data might not have a header. If the data is malformed (e.g., a single-row CSV), this could lead to incorrect results.Error Handling: The <code>float(row[index])</code> conversion can fail if any value in the column is not numeric. The script does not handle this case, which can cause a runtime exception.Empty Data: There is no check for an empty column or a situation where the column might not contain any numeric data, which can result in a division by zero when calculating the average.main() function:<ul style="list-style-type: none">Data Validity: There is no check to see if the <code>data</code> returned from <code>read_csv()</code> is empty or malformed. If the file is missing or empty, the script will fail in the <code>summarize_column()</code> function.Hardcoded File Path: The file path is hardcoded, which makes the script less flexible. It would be better to allow the user to provide the file path as input.Potential Index Error: In <code>summarize_column(data, 1)</code>, the index 1 is hardcoded, which assumes the relevant column for summarization is always the second column. This may cause an index error if the CSV structure changes. <p>Refactored Code</p> <pre>import csv import os def read_csv(filepath): """ Reads a CSV file and returns a list of rows. Handles errors like file not found. """ if not os.path.exists(filepath): print("Error: The file at {filepath} does not exist.") return [] try: with open(filepath, 'r') as f: reader = csv.reader(f) except csv.Error as e: print(f"Error reading the CSV file: {e}") return [] def summarize_column(data, index): """ Summarizes a specific column (skipping header). Returns the total and average of numeric values. """ if len(data) <= 1: print("Error: The file contains no data to summarize.") return 0, 0 values = [] for row in data[1:]: # skip header row try: value = float(row[index]) values.append(value) except (ValueError, IndexError): print(f"Warning: Non-numeric or missing value found at row {data.index(row)} in column {index}. Skipping.") continue if not values: print("Warning: No valid numeric data found in the column.") return 0, 0 total = sum(values) avg = total / len(values) return total, avg def main(): """ Main function to read the CSV, process data, and print summary. """ filepath = input("Enter the path to the CSV file: ").strip() data = read_csv(filepath) if not data: return # Exit if no data was loaded try: column_index = int(input("Enter the column index to summarize (starting from 0:)")) except ValueError: print("Error: Invalid column index.") return total, avg = summarize_column(data, column_index) if total != 0 or avg != 0: print("Total:", total) print("Average:", avg) else: print("No valid data to summarize.") if __name__ == '__main__': main()</pre>	<pre>#!/usr/bin/env python3 # This script is to parse and summarize numeric CSV files import csv import os from typing import List, Tuple, Optional def read_csv(filepath: str) -> List[List[str]]: """ Reads a CSV file and returns the data as a list of lists, with each inner list representing a row from the CSV. Handles file not found and general IO errors. Args: filepath: The path to the CSV file. Returns: A list of lists representing the CSV data. Returns an empty list if the file is empty. Raises: FileNotFoundError: If the file does not exist. (This is already raised by open(), but we document it) IOError: If an error occurs while reading the file. """ if not os.path.exists(filepath): raise FileNotFoundError(f"File not found at: {filepath}") try: with open(filepath, 'r', encoding='utf-8') as f: # Specify encoding reader = csv.reader(f) data = list(reader) # Read all rows into a list return data except IOError as e: raise IOError(f"Error reading file: {filepath} - {e}") except Exception as e: raise IOError(f"Unexpected error reading file: {filepath} - {e}") def summarize_column(data: List[List[str]], index: int) -> Tuple[float, float]: """ Summarizes a single column from the CSV data, calculating the total and average of the numeric values in the specified column. Args: data: A list of lists representing the CSV data. index: The index of the column to summarize (0-based). Returns: A tuple containing the total and average (float, float). Returns (0.0, 0.0) if the column has no valid numeric data. Raises: TypeError: If the data is not a list of lists, or if index is not an integer. IndexError: If the index is out of bounds for any row in the data. ValueError: If a value in the specified column cannot be converted to a float. """ if not isinstance(data, list) or not all(isinstance(row, list) for row in data): raise TypeError("Data must be a list of lists.") if not isinstance(index, int): raise TypeError("Index must be an integer.") if not data: # Handle empty data case return 0.0, 0.0 # Check for consistent row lengths and valid index "before" processing max_row_len = max(len(row) for row in data) if index < 0 or index >= max_row_len: raise IndexError(f"Index {index} is out of bounds for the data (max index is {max_row_len-1})") for row in data: if index >= len(row): raise IndexError(f"Index {index} is out of bounds for a row in the data.") values: List[float] = [] for row in data[1:]: # Skip header row "after" checking index bounds try: value = float(row[index]) values.append(value) except ValueError: raise ValueError(f"Non-numeric value found in column {index}: '{row[index]}'") if not values: # Handle the case where the column contains no valid numbers return 0.0, 0.0 total = sum(values) average = total / len(values) return total, average def main() -> None: """ Main function to execute the script. Reads data from 'data.csv', summarizes the second column (index 1), and prints the total and average. Handles potential errors during file processing and data summarization. """ filepath = 'data.csv' try: data = read_csv(filepath) total, average = summarize_column(data, 1) print(f"Total: {total}") print(f"Average: {average}") except (FileNotFoundError, IOError, TypeError, IndexError, ValueError) as e: print(f"Error: {e}") # Print the error message # Optionally, you could log the error or take other actions return # Exit the program more gracefully if __name__ == '__main__': main()</pre>
Zero-shot	<p>Analyze the following utility script. Identify any potential logic issues or design flaws. Then, refactor it for better readability, correctness, and safety.</p> <p>[code can be found in RawData.pdf]</p>	<pre>import csv import os def read_csv(filepath): """ Reads a CSV file and returns a list of rows. Handles errors like file not found. """</pre>	<pre>#!/usr/bin/env python3 # This script is to parse and summarize numeric CSV files import csv import os # For file existence check from typing import List, Tuple, Optional # For type hinting def read_csv(filepath: str) -> List[List[str]]: """</pre>

		<pre> if not os.path.exists(filepath): print("Error: The file at {filepath} does not exist.") return [] try: with open(filepath, 'r') as f: return [row for row in csv.reader(f)] except csv.Error as e: print("Error reading the CSV file: {e}") return [] def summarize_column(data, index): """ Summarizes a specific column (skipping header). Returns the total and average of numeric values. """ if len(data) <= 1: print("Error: The file contains no data to summarize.") return 0, 0 values = [] for row in data[1:]: # skip header row try: value = float(row[index]) values.append(value) except (ValueError, IndexError): print("Warning: Non-numeric or missing value found at row {data.index(row)} in column {index}. Skipping.") continue if not values: print("Warning: No valid numeric data found in the column.") return 0, 0 total = sum(values) avg = total / len(values) return total, avg def main(): """ Main function to read the CSV, process data, and print summary. """ filepath = input("Enter the path to the CSV file: ").strip() data = read_csv(filepath) if not data: return # Exit if no data was loaded try: column_index = int(input("Enter the column index to summarize (starting from 0): ")) except ValueError: print("Error: Invalid column index.") return total, avg = summarize_column(data, column_index) if total != 0 or avg != 0: print("Total:", total) print("Average:", avg) else: print("No valid data to summarize.") if __name__ == '__main__': main() </pre>	<pre> Reads a CSV file and returns the data as a list of lists. Args: filepath: The path to the CSV file. Returns: A list of lists, where each inner list represents a row of the CSV data. Returns an empty list if the file is empty. Raises: FileNotFoundError: If the file does not exist. IOError: If an error occurs while reading the file. """ if not os.path.exists(filepath): raise FileNotFoundError(f"File not found at {filepath}") try: with open(filepath, 'r', encoding='utf-8') as f: # Specify encoding reader = csv.reader(f) data = [list(str)] = list(reader) # Read all rows return data except IOError as e: raise IOError(f"Error reading CSV file: {e}") def summarize_column(data: List[List[str]], index: int) -> Tuple[float, float]: """ Calculates the total and average of the numeric values in a specified column of the CSV data. Args: data: A list of lists representing the CSV data. index: The index of the column to summarize (0-based). Returns: A tuple containing the total and average (float, float). Returns (0.0, 0.0) if the column contains no valid numeric data. Raises: TypeError: If data is not a list of lists or index is not an int. IndexError: If the index is out of bounds. ValueError: If a non-numeric value is encountered in the column. """ if not isinstance(data, list) or not all(isinstance(row, list) for row in data): raise TypeError("Input 'data' must be a list of lists.") if not isinstance(index, int): raise TypeError("Input 'index' must be an integer.") if not data: return 0.0, 0.0 # Handle empty data # Check for valid index before processing rows if index < 0 or any(index >= len(row) for row in data): raise IndexError(f"Index {index} is out of bounds for the data.") values = List[float] = [] for row in data[1:]: # Skip header row try: value = float(row[index]) values.append(value) except ValueError: raise ValueError(f"Non-numeric value in column {index}: '{row[index]}'") if not values: return 0.0, 0.0 # Handle empty column total = sum(values) average = total / len(values) return total, average def main() -> None: """ Main function to execute the script. Reads CSV data from 'data.csv', summarizes the second column, and prints the results. Includes error handling. """ filepath = 'data.csv' try: data = read_csv(filepath) total, average = summarize_column(data, 1) # Summarize the second column (index 1) print(f"Total: {total}") print(f"Average: {average}") except (FileNotFoundError, IOError, TypeError, IndexError, ValueError) as e: print(f"Error: {e}") # Print error message and exit return # Potential further processing of the data here if __name__ == '__main__': main() </pre>
--	--	---	---

This task involves a full analysis of given Python code, including debugging and refactoring. Chat-GPT's and Gemini's responses for both prompting methods are identical. Gemini's outputs contain more documentation, but functionally both work the same.

BONUS Code Completion and Robustness Enhancement (Python)

Chain-of-thought	<p>Complete the following file-processing script. The goal is to clean each line, remove punctuation, and count word frequencies correctly. Do it step by step:</p> <p>[code can be found in RawData.pdf]</p>	<pre> import string def load_file(filepath): """ Reads the file and returns the lines as a list of strings. """ with open(filepath, 'r') as f: return f.readlines() def clean_line(line): """ Cleans the input line by removing punctuation and converting to lowercase. """ # Remove punctuation using str.translate() and string.punctuation translator = str.maketrans("", "", string.punctuation) clean = line.translate(translator).lower() return clean def count_words(lines): """ Counts the frequency of each word in the provided lines. """ word_counts = {} for line in lines: clean = clean_line(line) for word in clean.split(): word_counts[word] = word_counts.get(word, 0) + 1 return word_counts </pre>	<pre> # file_processor.py - Script to process text files import string import os # For file existence check from typing import List, Dict # For type hinting def load_file(filepath: str) -> List[str]: """ Loads the lines from a text file. Args: filepath: The path to the text file. Returns: A list of strings, where each string is a line from the file. Returns an empty list if the file is empty or doesn't exist. Raises: FileNotFoundError: If the file does not exist. IOError: If an error occurs while reading the file. """ if not os.path.exists(filepath): raise FileNotFoundError(f"File not found at: {filepath}") try: with open(filepath, 'r', encoding='utf-8') as f: lines = List[str] = f.readlines() return lines except IOError as e: raise IOError(f"Error reading file: {filepath} - {e}") def clean_line(line: str) -> str: </pre>
------------------	--	--	--

		<pre> def main(): """ Main function to read file, process lines, count words, and print the results. """ filepath = 'input.txt' # Adjust the file path as needed lines = load_file(filepath) counts = count_words(lines) # Print the word counts sorted by word for word, count in sorted(counts.items()): print(f"{word}: {count}") if __name__ == '__main__': main() </pre>	<pre> """ Removes punctuation from a line of text and converts it to lowercase. Args: line: A string representing a line of text. Returns: A string with punctuation removed and converted to lowercase. Returns an empty string if the input line is empty. """ # Create a translation table to remove punctuation translator = str.maketrans('', '', string.punctuation) # Remove punctuation using the translation table cleaned_line = line.translate(translator) # Convert the line to lowercase cleaned_line = cleaned_line.lower() return cleaned_line def count_words(lines: List[str]) -> Dict[str, int]: """ Counts the frequency of each word in a list of lines. Args: lines: A list of strings, where each string is a line of text. Returns: A dictionary where keys are unique words and values are their counts. Returns an empty dictionary if the input list is empty. """ word_counts: Dict[str, int] = {} for line in lines: cleaned_line = clean_line(line) for word in cleaned_line.split(): word_counts[word] = word_counts.get(word, 0) + 1 return word_counts def main() -> None: """ Main function to execute the script. Loads text from 'input.txt', counts the word frequencies, and prints the results in sorted order. Handles potential errors during file processing. """ filepath = 'input.txt' try: lines = load_file(filepath) counts = count_words(lines) for word, count in sorted(counts.items()): print(f"{word}: {count}") except (FileNotFoundError, IOError) as e: print(f"Error: {e}") # Print the error message return # Exit the program if __name__ == '__main__': main() </pre>
Zero-shot	<p>Complete the following file-processing script. The goal is to clean each line, remove punctuation, and count word frequencies correctly.</p> <p>[code can be found in RawData.pdf]</p>	<pre> import string def load_file(filepath): """ Reads the file and returns the lines as a list of strings. """ with open(filepath, 'r') as f: return f.readlines() def clean_line(line): """ Cleans the input line by removing punctuation and converting to lowercase. """ # Remove punctuation using str.translate() and string.punctuation translator = str.maketrans("", "", string.punctuation) clean = line.translate(translator).lower() return clean def count_words(lines): """ Counts the frequency of each word in the provided lines. """ word_counts = {} for line in lines: clean = clean_line(line) # Clean the line by removing punctuation and making it lowercase for word in clean.split(): # Split the cleaned line into words word_counts[word] = word_counts.get(word, 0) + 1 # Count the words return word_counts def main(): """ Main function to read file, process lines, count words, and print the results. """ filepath = 'input.txt' # Adjust the file path as needed lines = load_file(filepath) counts = count_words(lines) # Print the word counts sorted by word for word, count in sorted(counts.items()): print(f"{word}: {count}") if __name__ == '__main__': main() </pre>	<pre> # file_processor.py - Script for processing text files import string import os from typing import List, Dict def load_file(filepath: str) -> List[str]: """ Loads the lines from a text file. Args: filepath: The path to the text file. Returns: A list of strings, where each string is a line from the file. Returns an empty list if the file is empty or doesn't exist. Raises: FileNotFoundError: If the file does not exist. IOError: If an error occurs while reading the file. """ if not os.path.exists(filepath): raise FileNotFoundError(f"File not found at: {filepath}") try: with open(filepath, 'r', encoding='utf-8') as f: lines: List[str] = f.readlines() return lines except IOError as e: raise IOError(f"Error reading file: {filepath} - {e}") def clean_line(line: str) -> str: """ Removes punctuation from a line of text and converts it to lowercase. Args: line: A string representing a line of text. Returns: A string with punctuation removed and converted to lowercase. """ # Create a translation table to remove punctuation translator = str.maketrans('', '', string.punctuation) cleaned_line = line.translate(translator) return cleaned_line.lower() def count_words(lines: List[str]) -> Dict[str, int]: """ Counts the frequency of each word in a list of lines. Args: lines: A list of strings, where each string is a line of text. Returns: A dictionary where keys are unique words and values are their counts. """ word_counts: Dict[str, int] = {} for line in lines: cleaned_line = clean_line(line) for word in cleaned_line.split(): word_counts[word] = word_counts.get(word, 0) + 1 return word_counts def main() -> None: """ Main function to execute the script. Loads text from 'input.txt', counts the word frequencies, and prints the results in sorted order. Handles potential file errors. """ filepath = 'input.txt' try: lines = load_file(filepath) counts = count_words(lines) </pre>

			<pre>for word, count in sorted(counts.items()): print(f"{word}: {count}") except (FileNotFoundError, IOError) as e: print(f"Error: {e}") if __name__ == '__main__': main()</pre>
--	--	--	---

This task involves cleaning and adding to given Python code. Chat-GPT's output for zero-shot prompting contains more documentation than its response for chain-of-thought, but functionally they are the same. Gemini's responses have the opposite pattern - there are less comments in the zero-shot response. Gemini's responses contain type hints, which enhances the code, and more documentation than Chat-GPT's responses.