GenAI for Software Development: Assignment 3
Joshua Pomeroy
jmpomeroy@wm.edu

Code Summarization (Java):

Gemini-Gemini BLEU score: 11.99
GPT-GPT BLEU score: 10.89
Gemini1-GPT1 BLEU score: 4.76
Gemini2-GPT2 BLEU score: 13.84

Zero-shot

Prompt	Gemini Response	GPT Response

Summarize the following Java method:
`public Map < String , Integer > countWordFrequency ( List < String > words ) {`
`Map < String , Integer > freqMap = new HashMap < >() ;`
`for ( String word : words ) {`
`freqMap . put ( word , freqMap . getOrDefault ( word , 0) + 1) ;`
`}`
`return freqMap ;`
`}`
	This Java method, countWordFrequency, takes a List of String objects as input and returns a Map where the keys are the unique words from the input list and the values are their corresponding frequencies (how many times each word appears in the list).
	
	Here's a breakdown of how it works:
	
	1. It initializes an empty HashMap called freqMap to store the word counts.
	2. It iterates through each word in the input words list.
	3. For each word, it checks if the word already exists as a key in the freqMap.
	If the word exists, it retrieves the current count using freqMap.get(word), increments it by 1, and updates the value associated with that word in the freqMap.
	If the word doesn't exist, freqMap.getOrDefault(word, 0) returns 0 (the default value), which is then incremented to 1, and the word is added to the freqMap with a frequency of 1.
	4. Finally, after processing all the words, the method returns the freqMap containing the word frequencies.
		The countWordFrequency method takes a list of strings (words) and returns a map (Map<String, Integer>) that represents the frequency of each word in the list.
		
		Summary:
		- It initializes an empty HashMap to store word-frequency pairs.
		- For each word in the list:
		- It increments the word's count in the map, using getOrDefault to handle words not yet in the map.
		- Finally, it returns the map containing each word and its corresponding count.

Role-Playing

Prompt	Gemini Response	GPT Response
You are an expert Java Software developer, summarize the following Java method: `public Map < String ,`	As a seasoned Java expert, I recognize this countWordFrequency method as a concise and efficient implementation for determining the occurrences of each unique word in a List<String>.	Method Summary: countWordFrequency This method computes the frequency of each word in a given list.

Integer > countWordFrequency (List < String > words) {
Map < String , Integer > freqMap = new HashMap < >() ;
for (String word : words) {
freqMap . put (word , freqMap . getOrDefault (word , 0) + 1) ;
}
return freqMap ;
}
	Internally, it employs a HashMap to store the word-frequency pairs. The method iterates through the input words list. For each word, it uses the getOrDefault method of the HashMap to retrieve the current count (or 0 if the word hasn't been encountered yet), increments it, and then updates the count for that word in the map. This approach avoids explicit checks for the existence of keys, resulting in cleaner and more performant code. The final Map<String, Integer> returned by the method provides a clear and readily usable representation of the frequency of each word in the original list. This is a standard and highly recommended pattern in Java for such tasks.	Functionality:
		- Initializes a HashMap<String, Integer> to store word counts.
		- Iterates over the input list words.
		- For each word:
		- Uses Map.getOrDefault() to retrieve the current count or default to 0 if the word is not present.
		- Increments the count and updates the map.
		- Returns the populated frequency map.
		
		Output:
		A Map<String, Integer> where each key is a unique word from the input list, and the corresponding value is the number of times that word appears.
		
		Performance:
		- Time Complexity: O(n), where n is the number of elements in the input list.
		- Space Complexity: O(k), where k is the number of unique words.
		
		Use Case:
		Useful for text analysis, such as word counting, building term frequency models, or pre-processing steps in natural language processing (NLP) tasks.

Comparison and Analysis: Both prompting techniques on both models appear to be correct summaries of what the function does. The zero-shot prompt of Gemini goes quite in depth so the role-playing does not add much information. With GPT however, role-playing goes much more in depth and includes information such as time and space complexity as well as when the function might be used.

Bug Fixing (Python – Off-by-One):

Gemini-Gemini CodeBLEU score: 100
GPT-GPT CodeBLEU score: 80.75
Gemini1-GPT1 CodeBLEU score: 100
Gemini2-GPT2 CodeBLEU score: 80.75

Zero-shot

Prompt	Gemini Response	GPT Response

Identify and fix the off-by-one error in this Python function:
def sum_range (start , end) :
total = 0
for i in range (start , end) :
total += i
	def sum_range(start, end):
	total = 0
	for i in range(start, end + 1):
	total += i
	return total
		def sum_range(start, end):
		total = 0
		for i in range(start, end + 1):
		total += i
		return total

return total		

Role-Playing

Prompt	Gemini Response	GPT Response

You are an expert Python Programmer, Identify and fix the off-by-one error in this Python function:
def sum_range (start , end) :
total = 0
for i in range (start , end) :
total += i
return total
	def sum_range(start, end):
	total = 0
	for i in range(start, end + 1):
	total += i
	return total
		def sum_range(start, end):
		total = 0
		for i in range(start, end + 1): # include end
		total += i
		return total

Comparison and Analysis: In this task changing the prompting technique or model has no impact on the outcome of the task as it is such a simple task with very few solutions and room for interpretation.

Bug Classification (C++):

Gemini-Gemini BLEU score: 19.93
GPT-GPT BLEU score: 9.53
Gemini1-GPT1 BLEU score: 3.26
Gemini2-GPT2 BLEU score: 8.66

Zero-shot

Prompt	Gemini Response	GPT Response

Classify the type of bug in the following C++ function:
int * getArray (int size)
{
int arr [size]; // Warning : local array
return arr ; // Bug: returning pointer to local variable
}
	Ah, that's a classic! The bug in the getArray function is a dangling pointer or returning a pointer to a local variable.
	
	Here's why:
	
	1. Local Array: Inside the function, int arr[size]; declares a local array named arr. This array is allocated on the stack frame of the getArray function.
	
	2. Stack Unwinding: When the getArray function finishes executing, its stack frame is unwound. This means the memory allocated for the local variable arr is no longer guaranteed to be valid. It might be overwritten by subsequent function calls.
	
	3. Returning a Pointer: The function returns arr, which in this context decays to a pointer to the first
		The bug in the given C++ function is a "Dangling Pointer" bug. Here's the analysis:
		
		Bug Type: Dangling Pointer / Use After Scope
		
		Explanation:
		- arr is a local variable allocated on the stack.
		- Once getArray() returns, the stack frame for this function is destroyed.
		- Therefore, the pointer arr becomes dangling-it points to memory that is no longer valid.
		- Any attempt to use the

	element of the local array. 4. Dangling Pointer: The pointer returned by getArray now points to a memory location that is no longer considered "owned" by the caller. Accessing the memory pointed to by this dangling pointer leads to undefined behavior. This could manifest as seemingly working correctly sometimes, crashing at other times, or producing garbage values, making it a particularly insidious type of bug to debug. In essence, the function is trying to hand back a memory address that is no longer valid after the function exits. To fix this, you would typically allocate the array on the heap using new int[size] and ensure that the caller is responsible for deallocating the memory using delete[] when it's no longer needed.	returned pointer leads to undefined behavior, such as crashes or corrupted data.

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Classify the type of bug in the following C++ function step by step:
int * getArray (int size)
{
int arr [size]; // Warning : local array
return arr ; // Bug: returning pointer to local variable
}
	Okay, let's break down the bug in that getArray function step by step:
	
	Step 1: Identify the Variable Declaration
	
	Inside the getArray function, a local array named arr is declared. The size of this array is determined by the input parameter size.
	The crucial point here is the keyword "local." This means the array arr has automatic storage duration. Its lifetime is limited to the scope of the function getArray. It exists only while the function is being executed. The memory for this array is typically allocated on the stack.
	
	Step 2: Understand Stack Memory Management
	
	When a function is called, a block of memory called a stack frame is created for it on the program's call stack. This stack frame holds the function's local variables, parameters, and return address.
	When the function finishes executing, its stack frame is unwound or deallocated. This means the memory that was used to store the local variables within that frame is no longer guaranteed to hold the same values or even be considered valid for access.
	
	Step 3: Analyze the Return Statement
	
	The function returns the identifier arr. In C++, when an array name is used in most contexts (including as a return value), it decays into a pointer to its first element. So, return arr; is equivalent to returning a pointer of type int* that points to the beginning of the local array arr.
	
	Step 4: Connect Lifetime and the Returned Pointer
		Step-by-step Classification:
		
		Step 1: Understand the Declaration
		
		- This declares a local array named arr with size elements.
		- In C++, arrays declared this way inside functions are allocated on the stack.
		- The lifetime of arr is limited to the duration of the function call.
		
		Type: Stack allocation (automatic storage duration)
		
		Step 2: Analyze the Return Statement
		
		- Here, arr decays to a pointer to its first element (i.e., int*).
		- However, since arr was a local stack variable, this pointer will refer to memory that no longer exists once the function ends.
		
		Type: Returning pointer to a destroyed (out-of-scope) variable
		
		Step 3: Identify the Bug
		- After the function returns, arr's

	The function returns a pointer to a memory location that was part of its local stack frame. However, as explained in Step 2, this stack frame is deallocated once the getArray function finishes. Step 5: Identify the Bug: The Dangling Pointer The pointer returned by getArray now points to a memory region that is no longer "owned" by the caller. The memory might be reused for subsequent function calls. This pointer is now a dangling pointer. A dangling pointer is a pointer that does not point to a valid memory location. Attempting to dereference (access the value at the memory location pointed to by) this dangling pointer leads to undefined behavior. This can result in: The program appearing to work correctly (by chance, the memory hasn't been overwritten yet). The program crashing. The program reading garbage or incorrect data. Security vulnerabilities. Conclusion: Type of Bug Therefore, the bug in the getArray function is a dangling pointer resulting from returning a pointer to a local variable (which has automatic storage duration and its memory is no longer valid after the function returns).	memory is invalid. - Any access to the returned pointer leads to undefined behavior. Bug Type: Dangling Pointer – a pointer that refers to a memory location that has been deallocated. Step 4: Result of the Bug - Undefined behavior: reading from or writing to the returned pointer may cause: - Crashes - Memory corruption - Unexpected output Final Classification: > Bug Type: Dangling Pointer > Cause: Returning a pointer to a local stack-allocated array that goes out of scope after function return.

Comparison and Analysis: Again with the zero-shot Gemini goes quite in depth, however its chain-of-thought response includes more on how the memory is handled and what a stack frame is. GPT's zero-shot is much higher level and shorter while the chain-of-thought brings in more information although still less in depth than Gemini.

Generating Email Validators (Python + Regex):

Gemini-Gemini CodeBLEU score: 56.29
GPT-GPT CodeBLEU score: 58.48
Gemini1-GPT1 CodeBLEU score: 47.07
Gemini2-GPT2 CodeBLEU score: 17.83

Zero-shot

Prompt	Gemini Response	GPT Response
Complete the Python function using regex to validate basic email addresses: **def is_valid_email (email):** **# TODO : Complete using regex** **pass**	import re def is_valid_email(email): """ Validates basic email addresses using regex. Args: email: The email address string to validate. Returns: True if the email address matches the basic pattern, False otherwise. """	**import re** **def is_valid_email(email):** **# Basic regex for email validation** **pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'** **return re.match(pattern, email) is not None**

	pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" return bool(re.fullmatch(pattern, email))	

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Complete the Python function step by step, using regex to validate basic email addresses:
def is_valid_email (email):
# TODO : Complete using regex 	
pass
	import re
	
	def is_valid_email(email):
	"""
	Validates basic email addresses using regex, step by step.
	
	Args:
	email: The email address string to validate.
	
	Returns:
	True if the email address matches the basic pattern, False otherwise.
	"""
	# Step 1: Define the regular expression pattern for a basic email address.
	# This pattern checks for:
	# - One or more alphanumeric characters, periods, underscores, percentage signs, plus signs, or hyphens
	# at the beginning (local part).
	# - Followed by an "@" symbol.
	# - Followed by one or more alphanumeric characters, periods, or hyphens (domain name).
	# - Followed by a literal dot ".".
	# - Followed by two or more alphabetic characters at the end (top-level domain).
	pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
	
	# Step 2: Use the 're.fullmatch()' function to attempt to match the entire
	# email string against the defined pattern.
	# 're.fullmatch()' returns a match object if the entire string matches the pattern,
	# and None otherwise.
	match = re.fullmatch(pattern, email)
	
	# Step 3: Check if a match object was returned.
	# If 'match' is not None, it means the entire email string conforms to the pattern.
	if match:
	return True
	else:
	return False
	
	# Example usage:
	print(is_valid_email("test@example.com"))
	print(is_valid_email("user.name@subdomain.example.co.uk"))
	print(is_valid_email("invalid-email"))
	print(is_valid_email("another@invalid"))
	print(is_valid_email("@missinglocal.com"))
	print(is_valid_email("local.missingat"))
		import re
		
		def is_valid_email(email):
		pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
		match = re.match(pattern, email)
		return match is not None

Comparison and Analysis: Both zero-shot and chain-of-thoughts appear to be correct for both models, however the code itself contains more comments with Gemini and less with GPT. This is most likely because in chain-of-thoughts, the text that the model produces that goes along with the code explains what the code does so GPT doesn't include the comments in the code while Gemini goes step by step in the code.

Generating Flask APIs (Python):

Gemini-Gemini CodeBLEU score: 92.52
GPT-GPT CodeBLEU score: 100
Gemini1-GPT1 CodeBLEU score: 64.37
Gemini2-GPT2 CodeBLEU score: 61.89

Zero-shot

Prompt	Gemini Response	GPT Response

Create a '/greet/<username>' endpoint that returns a JSON greeting:
from flask import Flask , jsonify
app = Flask (__name__)
@app . route ('/ greet / < username >')
def greet (username) :
# TODO : Return a JSON greeting 	
pass
	from flask import Flask, jsonify
	
	app = Flask(__name__)
	
	@app.route('/greet/<username>')
	def greet(username):
	greeting = {"message": f"Hello, {username}! Welcome!"}
	return jsonify(greeting)
	
	if __name__ == '__main__':
	app.run(debug=True)
		from flask import Flask, jsonify
		
		app = Flask(__name__)
		
		@app.route('/greet/<username>')
		def greet(username):
		return jsonify({'message': f'Hello, {username}!'})
		
		if __name__ == '__main__':
		app.run(debug=True)

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Step by step, create a '/greet/<username>' endpoint that returns a JSON greeting:
from flask import Flask , jsonify
app = Flask (__name__)
@app . route ('/ greet / < username >')
def greet (username) :
# TODO : Return a JSON greeting 	
pass
	from flask import Flask, jsonify
	
	app = Flask(__name__)
	
	@app.route('/greet/<username>')
	def greet(username):
	greeting = {"message": f"Hello, {username}! Welcome to Williamsburg!"}
	return jsonify(greeting)
	
	if __name__ == '__main__':
	app.run(debug=True)
		from flask import Flask, jsonify
		
		app = Flask(__name__)
		
		@app.route('/greet/<username>')
		def greet(username):
		return jsonify({'message': f'Hello, {username}!'})
		
		if __name__ == '__main__':
		app.run(debug=True)

Comparison and Analysis: Both zero-shot and chain-of-thoughts are almost identical in the final code produced. With chain-of-thoughts however, the natural language that comes along with the code is more in depth, which can be seen in the raw outputs.

SQL Schema Design (SQL):

Gemini-Gemini BLEU score: 100
GPT-GPT BLEU score: 56.24
Gemini1-GPT1 BLEU score: 48.46
Gemini2-GPT2 BLEU score: 34.91

Zero-shot

Prompt	Gemini Response	GPT Response
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)	CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL); CREATE TABLE books (id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT NOT NULL); CREATE TABLE reviews (id INTEGER PRIMARY KEY AUTOINCREMENT, user_id INTEGER NOT NULL, book_id INTEGER NOT NULL, rating INTEGER NOT NULL, FOREIGN KEY (user_id) REFERENCES users(id), FOREIGN KEY (book_id) REFERENCES books(id), CHECK (rating >= 1 AND rating <= 5));	-- Users Table CREATE TABLE users (id SERIAL PRIMARY KEY, name VARCHAR(100) NOT NULL); -- Books Table CREATE TABLE books (id SERIAL PRIMARY KEY, title VARCHAR(255) NOT NULL); -- Reviews Table CREATE TABLE reviews (id SERIAL PRIMARY KEY, user_id INTEGER NOT NULL, book_id INTEGER NOT NULL, rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5), -- Foreign Keys FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE, FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE, -- Unique constraint to prevent duplicate reviews by same user on same book UNIQUE (user_id, book_id));

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response
Step by step, 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)	CREATE TABLE users (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL); CREATE TABLE books (id INTEGER PRIMARY KEY AUTOINCREMENT, title TEXT NOT NULL);	CREATE TABLE users (id SERIAL PRIMARY KEY, -- Unique identifier for each user name VARCHAR(100) NOT NULL -- Name of the user, required); CREATE TABLE books (id SERIAL PRIMARY KEY, -- Unique identifier for each book title VARCHAR(255) NOT NULL -- Title of the book, required); CREATE TABLE reviews (id SERIAL PRIMARY KEY, -- Unique identifier for each review

	**CREATE TABLE reviews (
	  id INTEGER PRIMARY KEY AUTOINCREMENT,
	  user_id INTEGER NOT NULL,
	  book_id INTEGER NOT NULL,
	  rating INTEGER NOT NULL,
	  FOREIGN KEY (user_id) REFERENCES users(id),
	  FOREIGN KEY (book_id) REFERENCES books(id),
	  CHECK (rating >= 1 AND rating <= 5)
	);**
		** user_id INTEGER NOT NULL, -- References the user who wrote the review
		  book_id INTEGER NOT NULL, -- References the book being reviewed
		  rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5), -- Must be 1-5
		
		  -- Foreign keys to maintain referential integrity
		  FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
		  FOREIGN KEY (book_id) REFERENCES books(id) ON DELETE CASCADE,
		
		  -- Prevents a user from reviewing the same book more than once
		  UNIQUE (user_id, book_id)
		);**

Comparison and Analysis: I'm not too familiar with SQL however for Gemini at least, there was no change in zero-shot vs chain-of-thoughts. The chain-of-thoughts prompt provides more information however on what the model is doing for someone who might not know SQL.

Null Dereference Detection (Java):

Gemini-Gemini BLEU score: 12.87
GPT-GPT BLEU score: 22.51
Gemini1-GPT1 BLEU score: 2.4
Gemini2-GPT2 BLEU score: 8.5

Zero-shot

Prompt	Gemini Response	GPT Response

**Identify any null dereference risk:
 	
  public int getLength (String s) {
  return s . length () ; // What if s is null ?
  }**
	**You've hit the nail on the head! There is indeed a null dereference risk in your getLength method.
	
	If the String object referenced by the variable s is null, attempting to call the length() method on it will result in a NullPointerException. This is a runtime exception that will crash your program if not handled.
	
	The Risk:
	
	- NullPointerException: When you try to access a member (like a method or field) of a null object, the Java Virtual Machine (JVM) throws a NullPointerException.**
		**Yes, this code has a null dereference risk.
		
		Problem:
		
		If s is null, calling s.length() will throw a NullPointerException at runtime.**

Role-Playing

Prompt	Gemini Response	GPT Response

You are a seasoned Java developer, identify any null dereference risk:
 	
public int getLength (String s) {
return s . length () ; // What if s is null ?
}
	Ah, yes, a classic concern for any seasoned Java developer! You've correctly spotted the potential for a NullPointerException right there.
	
	The line return s.length(); directly accesses the length() method of the String object referenced by s. If, at the time this method is called, the s variable holds a null reference (meaning it doesn't point to any actual String object in memory), attempting to dereference it by calling a method will lead to the dreaded NullPointerException.
		As a seasoned Java developer, I can confirm that this method contains a null dereference risk.
		
		Risk:
		If the parameter s is null, the method call s.length() will result in a NullPointerException at runtime. This is a classic example of null dereference — attempting to access a method or field on a null reference.

Comparison and Analysis: Both Gemini and GPT seem to output almost identical outputs for zero-shot and role-playing with just slightly different wording.

CSV Parser Variants (Python):

Gemini-Gemini CodeBLEU score: 9.69
GPT-GPT CodeBLEU score: 18.61
Gemini1-GPT1 CodeBLEU score: 13.48
Gemini2-GPT2 CodeBLEU score: 55.04

Zero-shot

Prompt	Gemini Response	GPT Response
Improve the parser to support quoted fields: **def parse_csv_line (line) :** **return line . split (’,’) # Incomplete : doesn’t handle quoted fields**	**import csv** **def parse_csv_line(line):** **"""Parses a CSV line, handling quoted fields correctly."""** **reader = csv.reader([line])** **for row in reader:** **return row** **return []** **# Example usage** **line1 = "field1,field2,\"field,with,commas\",field4"** **result1 = parse_csv_line(line1)** **print(f"Input: {line1}")** **print(f"Output: {result1}")** **line2 = "value1,\"another value\",last one"** **result2 = parse_csv_line(line2)** **print(f"Input: {line2}")** **print(f"Output: {result2}")**	**import csv** **from io import StringIO** **def parse_csv_line(line):** **# Use Python's built-in csv module for proper handling** **reader = csv.reader(StringIO(line), skipinitialspace=True)** **return next(reader)**

	line3 = "just,simple" result3 = parse_csv_line(line3) print(f"Input: {line3}") print(f"Output: {result3}")	

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response
Step by step, improve the parser to support quoted fields: **def parse_csv_line (line) :** **return line . split (',') # Incomplete : doesn't handle quoted fields**	**def parse_csv_line(line):** **fields = []** **current_field = ""** **in_quotes = False** **for char in line:** **if char == '"':** **in_quotes = not in_quotes** **elif char == ',' and not in_quotes:** **fields.append(current_field)** **current_field = ""** **else:** **if in_quotes or char != '"': # Only add if inside quotes or not a quote** **current_field += char** **fields.append(current_field)** **return fields**	**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] == '"':** **field += '"' # Add escaped quote** **i += 1** **else:** **in_quotes = not in_quotes** **elif char == ',' and not in_quotes:** **fields.append(field)** **field = ''** **else:** **field += char** **i += 1** **fields.append(field)** **# Optional cleanup: strip outer quotes and surrounding spaces** **def clean(f):** **f = f.strip()** **if f.startswith('"') and f.endswith('"'):** **f = f[1:-1].replace('""', '"')** **return f** **return [clean(f) for f in fields]**

Comparison and Analysis: For this task, with both Gemini and GPT, using the chain-of-thoughts prompting seems to have caused the models to generate a more in depth code solution by doing things manually instead of using libraries.

Data Class to API Conversion (Kotlin):

Gemini-Gemini BLEU score: 37.03
GPT-GPT BLEU score: 76.09
Gemini1-GPT1 BLEU score: 44.41
Gemini2-GPT2 BLEU score: 74.55

Zero-shot

Prompt	Gemini Response	GPT Response

Convert the data class to a REST API using Ktor:
 	
data class Product (val id : Int , val name : String , val price : Double)
// TODO : Create GET and POST endpoints using Ktor
	import io.ktor.http.*
	import io.ktor.server.application.*
	import io.ktor.server.engine.*
	import io.ktor.server.netty.*
	import io.ktor.server.request.*
	import io.ktor.server.response.*
	import io.ktor.server.routing.*
	import kotlinx.serialization.Serializable
	import kotlinx.serialization.json.Json
	
	@Serializable
	data class Product(val id: Int, val name: String, val price: Double)
	
	val products = mutableListOf<Product>(
	Product(1, "Laptop", 1200.00),
	Product(2, "Mouse", 25.00)
	)
	
	fun main() {
	embeddedServer(Netty, port = 8080) {
	routing {
	get("/products") {
	call.respond(HttpStatusCode.OK, products)
	}
	
	get("/products/{id}") {
	val id = call.parameters["id"]?.toIntOrNull()
	if (id == null) {
	call.respond(HttpStatusCode.BadRequest, "Invalid product ID")
	return@get
	}
	val product = products.find { it.id == id }
	if (product != null) {
	call.respond(HttpStatusCode.OK, product)
	} else {
	call.respond(HttpStatusCode.NotFound, "Product not found")
	}
	}
	
	post("/products") {
	try {
	val product = call.receive<Product>()
	products.add(product)
		import io.ktor.application.*
		import io.ktor.http.*
		import io.ktor.request.*
		import io.ktor.response.*
		import io.ktor.routing.*
		import io.ktor.server.engine.*
		import io.ktor.server.netty.*
		import kotlinx.serialization.Serializable
		import io.ktor.features.ContentNegotiation
		import io.ktor.serialization.json
		
		// 1. Product Data Class
		@Serializable
		data class Product(val id: Int, val name: String, val price: Double)
		
		// 2. In-memory product storage
		val products = mutableListOf<Product>()
		
		// 3. Main entry point
		fun main() {
		embeddedServer(Netty, port = 8080) {
		install(ContentNegotiation) {
		json() // Enable JSON support
		}
		routing {
		// GET all products
		get("/products") {
		call.respond(products)
		}
		
		// POST a new product
		post("/products") {
		val product =

	call.respond(HttpStatusCode.Created, product)
	} catch (e: Exception) {
	
	call.respond(HttpStatusCode.BadRequest, "Invalid product data")
	}
	}
	}
	}.start(wait = true)
	}
		call.receive<Product>()
		
		products.add(product)
		
		call.respond(HttpStatusCode.Created, product)
		}
		}
		}.start(wait = true)
		}

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Step by step, convert the data class to a REST API using Ktor:
 	
data class Product (val id : Int , val name : String , val price : Double)
// TODO : Create GET and POST endpoints using Ktor
	import kotlinx.serialization.Serializable
	import io.ktor.http.*
	import io.ktor.serialization.kotlinx.json.*
	import io.ktor.server.application.*
	import io.ktor.server.engine.*
	import io.ktor.server.netty.*
	import io.ktor.server.plugins.contentnegotiation.*
	import io.ktor.server.request.*
	import io.ktor.server.response.*
	import io.ktor.server.routing.*
	
	@Serializable
	data class Product(val id: Int, val name: String, val price: Double)
	
	val products = mutableListOf<Product>(
	Product(1, "Laptop", 1200.00),
	Product(2, "Mouse", 25.00)
	)
	
	fun main() {
	embeddedServer(Netty, port = 8080) {
	install(ContentNegotiation) {
	json()
	}
	routing {
	// Will add GET and POST endpoints here in the next steps
	}
	}.start(wait = true)
	}
		import io.ktor.server.engine.*
		import io.ktor.server.netty.*
		import io.ktor.server.application.*
		import io.ktor.server.plugins.contentnegotiation.*
		import io.ktor.serialization.kotlinx.json.*
		import io.ktor.server.response.*
		import io.ktor.server.request.*
		import io.ktor.server.routing.*
		import io.ktor.http.*
		import kotlinx.serialization.Serializable
		
		@Serializable
		data class Product(val id: Int, val name: String, val price: Double)
		
		val products = mutableListOf<Product>()
		
		fun main() {
		embeddedServer(Netty, port = 8080) {
		install(ContentNegotiation) {
		json()
		}
		routing {
		get("/products") {
		call.respond(products)
		}
		post("/products") {
		val product = call.receive<Product>()
		products.add(product)
		
		call.respond(HttpStatusCode.Created, product)
		}
		}
		}.start(wait = true)
		}

Comparison and Analysis: I'm not at all familiar with kotlin so I can't say if either model correctly performed the task for either prompt. Based on the lengths however, using chain-of-thoughts seems to have reduced the total amount of code.

Function Summarization (Python):

Gemini-Gemini BLEU score: 0
GPT-GPT BLEU score: 0.23
Gemini1-GPT1 BLEU score: 8.37
Gemini2-GPT2 BLEU score: 100

Zero-shot

Prompt	Gemini Response	GPT Response
Write a brief summary of the function: **def reverse_words (sentence) :** **return ' '. join (sentence . split () [:: -1])**	**That Python function reverse_words takes a string called sentence as input. It first splits the sentence into a list of individual words using the split() method (which by default splits on whitespace). Then, it uses slicing [::-1] to create a reversed copy of this list of words. Finally, it joins the reversed list of words back into a single string with spaces in between each word using the ' '.join() method.** **In essence, this function reverses the order of words within a sentence.**	**The function reverse_words(sentence) takes a string sentence as input, splits it into words, reverses the order of the words, and then joins them back into a single string separated by spaces. It returns this reversed word order as the output.**

One-shot

Prompt	Gemini Response	GPT Response
### Code: **def is_even(n):** **return n % 2 == 0** **### Summary:** **Checks if a number is even.** **### Code:** **def reverse_words (sentence) :** **return ' '. join (sentence . split () [:: -1])** **### Summary:**	**Reverses the order of words in a sentence.**	**Reverses the order of words in a sentence.**

Comparison and Analysis: In this task there is a large difference when using a better prompting technique than zero-shot. The zero-shot prompt is very verbose and not very brief. On the other hand, by using just one-shot with an example of a single sentence summary, both models are able to then generate a single sentence summary for the input function.

Prompt from Code Comments (Python):

Gemini-Gemini BLEU score: 0
GPT-GPT BLEU score: 42.44

Gemini1-GPT1 BLEU score: 1.4
Gemini2-GPT2 BLEU score: 5.89

Zero-shot

Prompt	Gemini Response	GPT Response

Write a prompt that could generate the following Python code:
 	
`# This function checks if a number is prime`
`def is_prime ( n ) :`
`  if n <= 1:`
`  return False`
`  for i in range (2 , int( n **0.5) +1) :`
`  if n % i == 0:`
`  return False`
`  return True`
	Here are a few prompt options that could generate the Python code you provided, varying slightly in their phrasing:
	
	Option 1 (Direct and Functional):
	
	> Write a Python function called is_prime that takes an integer n as input and returns True if n is a prime number and False otherwise. The function should include a check for numbers less than or equal to 1 and optimize the primality test by only checking divisibility up to the square root of n.
	
	Option 2 (Slightly More Descriptive):
	
	> Create a Python function that determines if a given integer is a prime number. The function should be named is_prime and accept a single integer argument. Ensure the function handles the base case where the number is less than or equal to 1. For efficiency, the divisibility check should only iterate up to the integer part of the square root of the input number.
	
	Option 3 (Focusing on the Logic):
	
	> Generate Python code for a function is_prime(n) that implements the standard primality test. This function should first handle the case of n being less than or equal to 1. Then, it should iterate through potential divisors from 2 up to the square root of n. If any of these divisors evenly divide n, the function should return False. Otherwise, if the loop completes without finding a divisor, the function should return True.
	
	Option 4 (Concise):
	
	> Write a Python function is_prime(n) that efficiently checks if an integer n is prime.
		Write a Python function called is_prime(n) that returns True if n is a prime number and False otherwise. The function should check if n is less than or equal to 1 (not prime), and for other values, it should check divisibility from 2 up to the square root of n. Include a comment explaining the purpose of the function.

	All of these prompts aim to guide the language model towards generating the specific Python code you provided by highlighting the key aspects of the function: its name, input, output, the base case for non-prime numbers, and the optimization using the square root.	

One-shot

Prompt	Gemini Response	GPT Response
### Code: **def is_even(n):** **return n % 2 == 0** **### Prompt:** **Write a Python function to check if a number is even** **### Code:** **# This function checks if a number is prime** **def is_prime (n) :** **if n <= 1:** **return False** **for i in range (2 , int(n **0.5) +1) :** **if n % i == 0:** **return False** **return True** **### Prompt:**	**Write a Python function to check if a number is prime, including handling edge cases and optimizing for efficiency.**	**Write a Python function called is_prime(n) that returns True if the input number n is a prime number and False otherwise. Include a comment describing the function's purpose, and use an efficient method that checks divisibility only up to the square root of n.**

Comparison and Analysis: In zero-shot, the goal is specified however it is also loose, as seen with Gemini generating 4 different prompts instead of a single one. With one-shot Gemini only creates one prompt which is much shorter and concise than any of the prompts generated in zero-shot. As for GPT, there does not seem to be much change.

Fixing Factorial Bug (Python):

Gemini-Gemini CodeBLEU score: 52.06
GPT-GPT CodeBLEU score: 100
Gemini1-GPT1 CodeBLEU score: 35.08
Gemini2-GPT2 CodeBLEU score: 67.63

Zero-shot

Prompt	Gemini Response	GPT Response
Fix the bug when input is 0: **def factorial (n) :** **result = 1**	**def factorial(n):** **if n < 0:** **return "Factorial is not defined for negative numbers"** **elif n == 0:** **return 1**	**def factorial(n):** **result = 1** **for i in range(1, n + 1):** **result *= i**

for i in range (1 , n) : result *= i return result	else: result = 1 for i in range(1, n + 1): # Changed the range to include n result *= i return result	return result

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Fix the bug step by step when input is 0:
 	
def factorial (n) :
result = 1
for i in range (1 , n) :
result *= i
return result
	def factorial(n):
	if n == 0:
	return 1
	else:
	result = 1
	for i in range(1, n + 1):
	result *= i
	return result
		def factorial(n):
		result = 1
		for i in range(1, n + 1):
		result *= i
		return result

Comparison and Analysis: Again like with a previous simple task, changing the prompting method only causes Gemini this time to remove the comments from the code. The code itself is the exact same for both models and prompting techniques.

Linked List Node Deletion (C):

Gemini-Gemini CodeBLEU score: 67.52
GPT-GPT CodeBLEU score: 90.17
Gemini1-GPT1 CodeBLEU score: 26.12
Gemini2-GPT2 CodeBLEU score: 21.32

Zero-shot

Prompt	Gemini Response	GPT Response

Implement node deletion by value:
 	
struct Node {
int data ;
struct Node * next ;
};
 	
void deleteNode (struct Node ** head , int key) {
// TODO : Implement node deletion
}
	#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;
	
	// Case 1: Empty list
	if (*head == NULL) {
	return;
	}
	
	// Case 2: Key is in the head node
	if (current != NULL && current->data == key) {
	*head = current->next; // Changed head
		#include <stdlib.h>
		
		struct Node {
		int data;
		struct Node *next;
		};
		
		void deleteNode(struct Node **head, int key) {
		struct Node *temp = *head;
		struct Node *prev = NULL;

```
        free(current);        // Free old head
        return;
    }


    // Case 3: Search for the key in the list
    while (current != NULL && current->data
!= key) {
        prev = current;
        current = current->next;
    }


    // Case 4: Key not found
    if (current == NULL) {
        return;
    }


    // Case 5: Key found - Unlink the node
from the list
    prev->next = current->next;


    // Free the memory of the deleted node
    free(current);
}


// Helper function to print the linked list (for
testing)
void printList(struct Node *n) {
    while (n != NULL) {
        printf("%d ", n->data);
        n = n->next;
    }
    printf("\n");
}


// Helper function to create a new node (for
testing)
struct Node* newNode(int data) {
    struct Node* new_node = (struct
Node*)malloc(sizeof(struct Node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}


int main() {
    // Example usage:
    struct Node* head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(3);
    head->next->next->next = newNode(4);


    printf("Original list: ");
    printList(head);


    deleteNode(&head, 3);
    printf("List after deleting 3: ");
    printList(head);


    deleteNode(&head, 1);
    printf("List after deleting 1: ");
    printList(head);


    deleteNode(&head, 5); // Key not present
    printf("List after deleting 5 (not found):
");
    printList(head);
```

```
    // Case 1: The head node
itself holds the key
    if (temp != NULL &&
temp->data == key) {
        *head = temp->next;  //
Move head to next node
        free(temp);          // Free
old head
        return;
    }


    // Case 2: Search for the
key
    while (temp != NULL &&
temp->data != key) {
        prev = temp;
        temp = temp->next;
    }


    // If key was not present
in the list
    if (temp == NULL) return;


    // Unlink the node from
the list
    prev->next = temp->next;
    free(temp);
}
```

	deleteNode(&head, 4);
	printf("List after deleting 4: ");
	printList(head);
	
	deleteNode(&head, 2);
	printf("List after deleting 2 (last remaining): ");
	printList(head);
	
	deleteNode(&head, 2); // Deleting from an empty list
	printf("List after deleting 2 from empty list: ");
	printList(head);
	
	return 0;
	}	

Two-shot

Prompt	Gemini Response	GPT Response

Original List: 3 2 5 7
List after deleting 3: 2 5 7
 	
Original List: 3 2 5 7
List after deleting 5: 3 2 7
 	
struct Node {
int data ;
struct Node * next ;
};
 	
void deleteNode (struct Node ** head , int key) {
// TODO : Implement node deletion
}
	#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;
	
	// Case 1: Empty list
	if (*head == NULL) {
	return;
	}
	
	// Case 2: Key is in the head node
	if (current != NULL && current->data == key) {
	*head = current->next; // Changed head
	free(current); // Free old head
	return;
	}
	
	// Case 3: Search for the key in the list
	while (current != NULL && current->data != key) {
	prev = current;
	current = current->next;
	}
	
	// Case 4: Key not found
	if (current == NULL) {
	return;
	}
	
	// Case 5: Key found - Unlink the node from the list
	prev->next = current->next;
	
	// Free the memory of the deleted node
	free(current);
	}
	
	// Helper function to print the linked list (for testing)
	void printList(struct Node *n) {
	while (n != NULL) {
	printf("%d ", n->data);
	n = n->next;
		#include <stdlib.h>
		
		struct Node {
		int data;
		struct Node *next;
		};
		
		void deleteNode(struct Node **head, int key) {
		struct Node *temp = *head;
		struct Node *prev = NULL;
		
		// Case 1: Head node holds the key
		if (temp != NULL && temp->data == key) {
		*head = temp->next; // Move head to the next node
		free(temp); // Free old head
		return;
		}
		
		// Case 2: Traverse the list to find the key
		while (temp != NULL &&

	}
	printf("\n");
	}
	
	// Helper function to create a new node (for testing)
	struct Node* newNode(int data) {
	struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
	new_node->data = data;
	new_node->next = NULL;
	return new_node;
	}
	
	// Helper function to create the linked list from an array (for testing)
	void createList(struct Node** head, int arr[], int n) {
	*head = NULL;
	struct Node* temp = NULL;
	
	for (int i = 0; i < n; i++) {
	struct Node* new_node = newNode(arr[i]);
	if (*head == NULL) {
	*head = new_node;
	temp = *head;
	} else {
	temp->next = new_node;
	temp = new_node;
	}
	}
	}
	
	int main() {
	// Example 1: Delete 3
	struct Node* head1 = NULL;
	int arr1[] = {3, 2, 5, 7};
	int n1 = sizeof(arr1) / sizeof(arr1[0]);
	createList(&head1, arr1, n1);
	
	printf("Original List: ");
	printList(head1);
	
	int keyToDelete1 = 3;
	deleteNode(&head1, keyToDelete1);
	printf("List after deleting %d: ", keyToDelete1);
	printList(head1);
	
	// Example 2: Delete 5
	struct Node* head2 = NULL;
	int arr2[] = {3, 2, 5, 7};
	int n2 = sizeof(arr2) / sizeof(arr2[0]);
	createList(&head2, arr2, n2);
	
	printf("\nOriginal List: ");
	printList(head2);
	
	int keyToDelete2 = 5;
	deleteNode(&head2, keyToDelete2);
	printf("List after deleting %d: ", keyToDelete2);
	printList(head2);
	
	// Remember to free the allocated memory in a real application
	// This simplified example omits explicit memory freeing for brevity in demonstration.
	
	return 0;
	}
		temp->data != key) {
		prev = temp;
		temp = temp->next;
		}
		
		// If key is not found, do nothing
		if (temp == NULL) return;
		
		// Unlink the node and free memory
		prev->next = temp->next;
		free(temp);
		}

Comparison and Analysis: With this task, Gemini generates example usages for both zero-shot and two-shot prompting. It also takes the liberty of generating helper functions to print the linked list and creating the list and nodes. On the other hand, GPT does not provide example usages for either technique. The code itself in the deletion method appears to be nearly identical.

Recursive Function Completion (Python):

Gemini-Gemini CodeBLEU score: 30.01
GPT-GPT CodeBLEU score: 95.22
Gemini1-GPT1 CodeBLEU score: 32.61
Gemini2-GPT2 CodeBLEU score: 91.19

Zero-shot

Prompt	Gemini Response	GPT Response
Complete the recursive function for Fibonacci: **def fibonacci (n) :** **# TODO : Base cases and recursive call** **pass**	**def fibonacci(n):** **"""** **Calculates the nth Fibonacci number recursively.** **Args:** **n: The index of the desired Fibonacci number (non-negative integer).** **Returns:** **The nth Fibonacci number.** **"""** **# Base cases** **if n == 0:** **return 0** **elif n == 1:** **return 1** **# Recursive call** **else:** **return fibonacci(n - 1) + fibonacci(n - 2)** **# Example Usage:** **print(f"fibonacci(0) = {fibonacci(0)}")** **print(f"fibonacci(1) = {fibonacci(1)}")** **print(f"fibonacci(2) = {fibonacci(2)}")** **print(f"fibonacci(3) = {fibonacci(3)}")** **print(f"fibonacci(4) = {fibonacci(4)}")** **print(f"fibonacci(5) = {fibonacci(5)}")** **print(f"fibonacci(10) = {fibonacci(10)}")**	**def fibonacci(n):** **# Base cases** **if n == 0:** **return 0** **elif n == 1:** **return 1** **# Recursive call** **return fibonacci(n - 1) + fibonacci(n - 2)**

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response
Step by step complete the recursive function for Fibonacci: **def fibonacci (n) :** **# TODO : Base cases and recursive call** **pass**	**def fibonacci(n):** **# Base cases** **if n == 0:** **return 0** **elif n == 1:** **return 1** **# Recursive call** **else:** **return fibonacci(n - 1) + fibonacci(n - 2)**	**def fibonacci(n):** **# Base cases** **if n == 0:** **return 0** **elif n == 1:** **return 1** **# Recursive case** **return fibonacci(n - 1) + fibonacci(n - 2)**

Comparison and Analysis: In this task, the zero-shot prompt generates a simple and working function. Gemini's zero-shot response is well commented and includes example usages of the function. When moving to chain-of-thoughts, this commenting and example usage is instead moved to natural language instead of comments in the code.

Constructor Completion (Python):

Gemini-Gemini CodeBLEU score: 56.43
GPT-GPT CodeBLEU score: 100
Gemini1-GPT1 CodeBLEU score: 56.43
Gemini2-GPT2 CodeBLEU score: 100

Zero-shot

Prompt	Gemini Response	GPT Response
Complete the class constructor: **class Person :** **def __init__ (self) :** **# TODO : Add name , age , and optional email** **pass**	**class Person:** **def __init__(self, name, age, email=None):** **"""** **Initializes a Person object.** **Args:** **name (str): The name of the person.** **age (int): The age of the person.** **email (str, optional): The email address of the person. Defaults to None.** **"""** **self.name = name** **self.age = age** **self.email = email**	**class Person:** **def __init__(self, name, age, email=None):** **self.name = name** **self.age = age** **self.email = email**

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response
Step by step complete the class constructor: **class Person :** **def __init__ (self) :** **# TODO : Add name , age , and optional email** **pass**	**class Person:** **def __init__(self, name, age, email=None):** **self.name = name** **self.age = age** **self.email = email**	**class Person:** **def __init__(self, name, age, email=None):** **self.name = name** **self.age = age** **self.email = email**

Comparison and Analysis: Again, Gemini well comments the code in zero-shot so that it can be easily understood. With chain-of-thoughts, Gemini explains the code in natural language so it most likely doesn't feel the need to explain the code again in the comments.

Binary Search Completion (Java):

Gemini-Gemini CodeBLEU score: 88.81

GPT-GPT CodeBLEU score: 100
Gemini1-GPT1 CodeBLEU score: 52.98
Gemini2-GPT2 CodeBLEU score: 49.42

Zero-shot

Prompt	Gemini Response	GPT Response

Complete the binary search implementation:
 	
public int binarySearch (int [] arr , int target) {
int left = 0 , right = arr . length - 1;
while (left <= right) {
int mid = (left + right) / 2;
// TODO : Compare and adjust bounds
}
return -1;
}
	public int binarySearch (int [] arr , int target) {
	int left = 0 , right = arr . length - 1;
	while (left <= right) {
	int mid = (left + right) / 2;
	if (arr [mid] == target) {
	return mid; // Target found 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 in the array
	}
		public int binarySearch(int[] arr, int target) {
		int left = 0, right = arr.length - 1;
		while (left <= right) {
		int mid = (left + right) / 2;
		
		if (arr[mid] == target) {
		return mid; // Target found
		} else if (arr[mid] < target) {
		left = mid + 1; // Search right half
		} else {
		right = mid - 1; // Search left half
		}
		}
		return -1; // Target not found
		}

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response

Step by step complete the binary search implementation:
 	
public int binarySearch (int [] arr , int target) {
int left = 0 , right = arr . length - 1;
while (left <= right) {
int mid = (left + right) / 2;
// TODO : Compare and adjust bounds
}
return -1;
}
	public int binarySearch (int [] arr , int target) {
	int left = 0 , right = arr . length - 1;
	while (left <= right) {
	int mid = (left + right) / 2;
	if (arr [mid] == target) {
	return mid; // Target found at the middle index.
	} else if (arr [mid] < target) {
	left = mid + 1; // Search in the right half.
	} else { // arr[mid] > target
	right = mid - 1; // Search in the left half.
	}
	}
	return -1;
	}
		public int binarySearch(int[] arr, int target) {
		int left = 0, right = arr.length - 1;
		while (left <= right) {
		int mid = (left + right) / 2;
		
		if (arr[mid] == target) {
		return mid; // Target found
		} else if (arr[mid] < target) {
		left = mid + 1; // Search right half
		} else {
		right = mid - 1; // Search left half
		}
		}
		return -1; // Target not found
		}

Comparison and Analysis: As shown in the CodeBLEU scores, there is almost no change for Gemini (the only change seems to be punctuation in the comments) and there is no change for GPT.

Self-Consistency Bug Fixing (C++):

Gemini-Gemini CodeBLEU score: 100
GPT-GPT CodeBLEU score: 100

Gemini1-GPT1 CodeBLEU score: 79.54
Gemini2-GPT2 CodeBLEU score: 79.54

Zero-shot

Prompt	Gemini Response	GPT Response

Resolve inconsistency between function name and logic:
 	
// Supposed to return true if x is even
bool isOdd (int x) {
return x % 2 == 0; // Logic contradicts function name
}
	bool isEven (int x) {
	return x % 2 == 0;
	}
		bool isEven(int x) {
		return x % 2 == 0;
		}

One-shot

Prompt	Gemini Response	GPT Response

Bad function:
bool isEven (int x) {
return x % 2 != 0; // or return x % 2 == 1;
}
Correct function:
bool isOdd (int x) {
return x % 2 != 0; // or return x % 2 == 1;
}
 	
Bad Function:
bool isOdd (int x) {
return x % 2 == 0;
}
Correct Function:
	bool isEven (int x) {
	return x % 2 == 0;
	}
		bool isEven(int x) {
		return x % 2 == 0;
		}

Comparison and Analysis: With this simple task, changing the prompting technique has no impact on the final outcome. This is most likely due to the fact that there is really only one solution to this task, so changing the prompt will not result in different code.

Prompt Chaining: Bug Identification → Fix (JavaScript):

Gemini-Gemini CodeBLEU score: 92.15
GPT-GPT CodeBLEU score: 100
Gemini1-GPT1 CodeBLEU score: 68.62
Gemini2-GPT2 CodeBLEU score: 61.49

Zero-shot

Prompt	Gemini Response	GPT Response

Identify and fix the bug:
 	
function isEven (n) {
return n % 2; // Returns 1 or 0 , not true / false
}
	function isEven(n) {
	return n % 2 === 0; // Returns true if the remainder is 0, false otherwise
	}
		function isEven(n) {
		return n % 2 === 0;
		}

Prompt Chaining

Prompt 1	Gemini Response 1	GPT Response 1

Only identify the bug:
 	
function isEven (n) {
return n % 2; // Returns 1 or 0 , not true / false
}
	The bug is that the function returns the remainder of the division by 2 (which will be 0 or 1), instead of a boolean value (true or false).	The bug is that the function returns 1 or 0 instead of true or false.

Prompt 2	Gemini Response 2	GPT Response 2

Now fix the bug	function isEven (n) {
	return n % 2 === 0; // Returns true if the remainder is 0, false otherwise
	}
		function isEven(n) {
		return n % 2 === 0;
		}

Comparison and Analysis: In this task I try prompt chaining as the secondary technique. The task has two components to it resulting in an easy split to chain prompts. However, even when chaining the prompts, there is no change at all in the final outcome of code, the bug is correctly fixed in both scenarios.

Summary Decomposition (C++):

Gemini-Gemini BLEU score: 14.57
GPT-GPT BLEU score: 30.4
Gemini1-GPT1 BLEU score: 13.75
Gemini2-GPT2 BLEU score: 5.95

Zero-shot

Prompt	Gemini Response	GPT Response

Decompose the high-level comment/summary into logical steps:
 	
// Function that validates an input , calculates square , and returns result
int process (int x) {
if (x < 0) return -1;
return x * x ;
}
	Okay, I can break down that high-level comment into logical steps based on the code:
	
	1. Validate Input: The function first checks if the input x meets a certain condition.
	2. Conditional Return (Invalid): If the input x is less than 0, the function immediately returns -1, indicating an invalid input.
	3. Calculate Square (Valid): If the input x is not less than 0 (meaning it's valid according to the condition), the function proceeds to calculate the square of x.
	4. Return Result (Valid): The function then returns the calculated square of x.
		1. Validate the input:
		Check if the input x is less than 0. If so, return -1 to indicate invalid input.
		
		2. Calculate the square:
		If the input is valid, compute the square of x (i.e., x * x).
		
		3. Return the result:
		Return the squared value as the final result.

Chain-of-Thoughts

Prompt	Gemini Response	GPT Response
Step by step, decompose the high-level comment/summary into logical steps: **// Function that validates an input , calculates square , and returns result** **int process (int x) {** **if (x < 0) return -1;** **return x * x ;** **}**	**Alright, let's break down that comment and code step by step:** **1. Function Definition: The code defines a function named process that takes one integer argument, which is named x. The function is declared to return an integer value (int).** **2. Input Validation (Conditional Check): Inside the process function, there's an if statement that checks a condition: "is x less than 0?".** **3. Return Invalid Indicator: If the condition in the if statement is true (i.e., if x is indeed less than 0), the function immediately executes the statement return -1;. This indicates that the input was not valid according to the function's criteria.** **4. Calculate Square (Alternative Path): If the condition in the if statement is false (i.e., if x is not less than 0), the code execution skips the return -1; statement and proceeds to the next line. This line calculates the square of x by multiplying x by itself (x * x).** **5. Return Calculated Result: Finally, the function executes the statement return x * x;, which returns the calculated square of the input x. This happens only when the initial validation check (x < 0) fails.** **So, in essence, the function first checks if the input is non-negative. If it is, it computes and returns the square; otherwise, it returns a specific value (-1) to signal an invalid input.**	**High-Level Summary: "Function that validates an input, calculates square, and returns result"** **Step-by-Step Decomposition:** **1. Input Validation** **- Check if the input x is a negative number (x < 0).** **- If the input is invalid (i.e., negative), return -1 to indicate an error or special condition.** **2. Calculation** **- If the input passes validation (i.e., it is non-negative), compute the square of x using x * x.** **3. Return the Result** **- Return the squared value as the final result of the function.**

Comparison and Analysis: In this task, for the zero-shot prompting, it appears that both models generated a chain-of-thoughts like response. This is most likely due to the original prompt including the word “steps” resulting in the models creating the step format in zero-shot. Gemini however has one change in the chain-of-thoughts prompt, it includes the function definition in its breakdown of the function.

Purpose Inference → Completion (Python):

Gemini-Gemini CodeBLEU score: 100
GPT-GPT CodeBLEU score: 87.34
Gemini1-GPT1 CodeBLEU score: 49.83
Gemini2-GPT2 CodeBLEU score: 49.17

Zero-shot

Prompt	Gemini Response	GPT Response

Complete the function based on intent:
 	
def calculate_average (scores) :
total = 0
# TODO : Complete to return average 	
pass
	def calculate_average (scores) :
	total = 0
	if not scores:
	return 0
	for score in scores:
	total += score
	average = total / len(scores)
	return average
		def calculate_average(scores):
		total = 0
		for score in scores:
		total += score
		if len(scores) == 0:
		return 0 # Avoid division by zero
		return total / len(scores)

Two-shot

Prompt	Gemini Response	GPT Response

Input: 3, 10, 5, 11, 4
Output: 6.6
 	
Input: 5, 12, 52, 2, 4
Output: 15
 	
def calculate_average (scores) :
total = 0
	def calculate_average (scores) :
	total = 0
	if not scores:
	return 0
	for score in scores:
	total += score
	average = total / len(scores)
	return average
		def calculate_average(scores):
		total = 0
		for score in scores:
		total += score
		if len(scores) == 0:
		return 0 # or raise an exception, depending on needs
		return total / len(scores)

Comparison and Analysis: In this simple task, the zero-shot and two-shot prompt had no impact on the code produced. The only change is in the wording of the comments produced by GPT.

Summary

It appears for simple tasks or tasks with little interpretations, changing the prompting technique or even model, has little to no effect on the end outcome. Tasks such as fixing an explicit bug or creating a simple function are trivial for the model regardless of the prompt. Overall, Gemini appears to usually go more in depth than GPT with comments and explanations when using zero-shot prompting. GPT improves its level of depth when applying better techniques such as few-shot or chain-of-thought.