Lab: Get Familiar with GenAI Classroom

Estimated Time: 20 mins

Introduction:

Within the Generative AI Classroom, you have the opportunity to delve into the art of prompt engineering. Here, you can actively engage in writing and comparing your prompts, allowing you to refine your skills through real-time chat responses.Additionally, you'll have the option to utilize the GPT-5 Nano model, an upgraded version of OpenAI's language model. This enhanced iteration is specifically designed to significantly enhance its ability in generating text that closely resembles human language while comprehending and adapting to various tasks and situations.

Learning Objectives:

After completing this lab, you should be able to perform the following tasks:

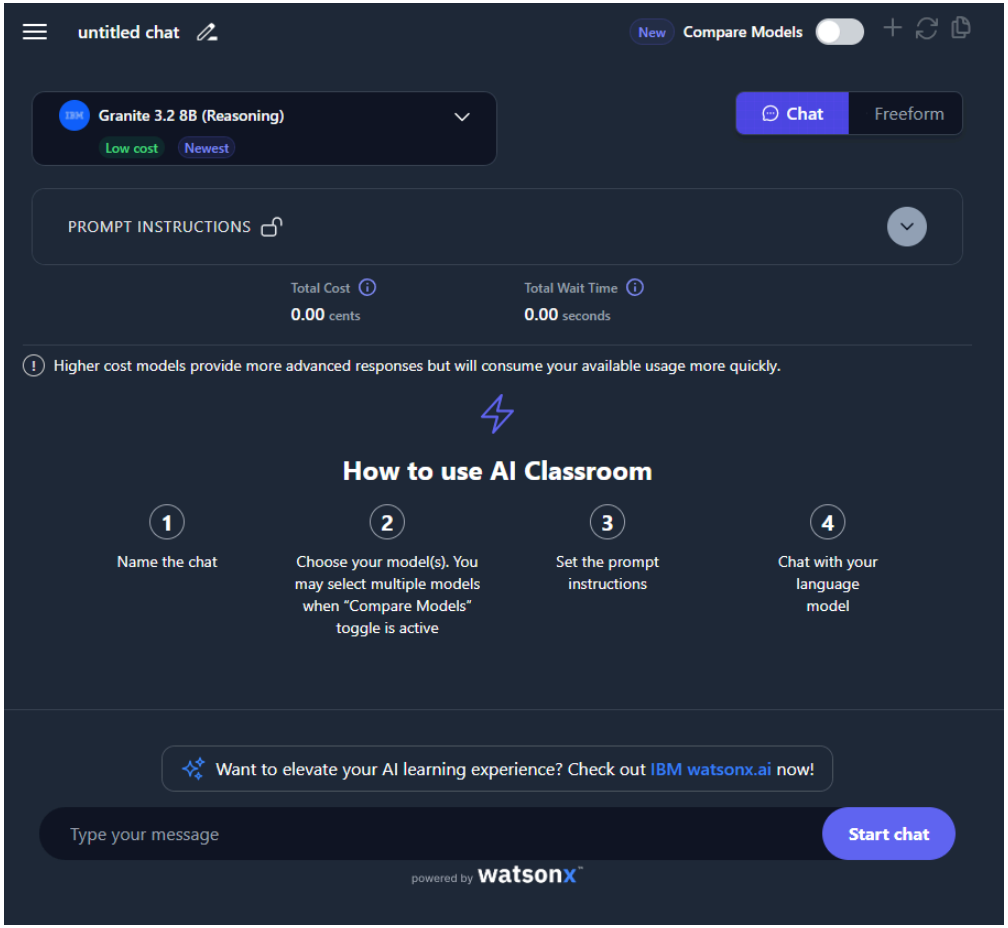
- Develop a working understanding of the GenAI Classroom platform, exploring its features and tools tailored for learning prompt engineering.
- Engage in a conversation with the chatbot, asking various questions to observe how it responds accurately, showcasing the functionality of these prompts.

  Please note that generative AI is an evolving field. As you attempt the labs, your experience and output might be different than what is seen here.

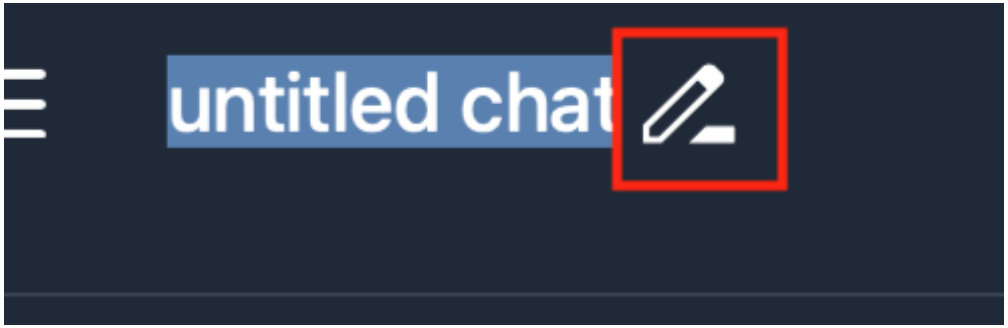
Task 1: Set up your AI Classroom

Within a lab environment, you will have a single-page display featuring both the Generative AI Classroom interface and your instructional content. The Instructions will be positioned on the left-hand side, while the Generative AI Classroom environment will be accessible on the right-hand side of the same page. This layout facilitates simultaneous access to guidance and the interactive learning environment, enhancing convenience and ease of use.

1. The desired appearance of your AI Classroom should reflect this.

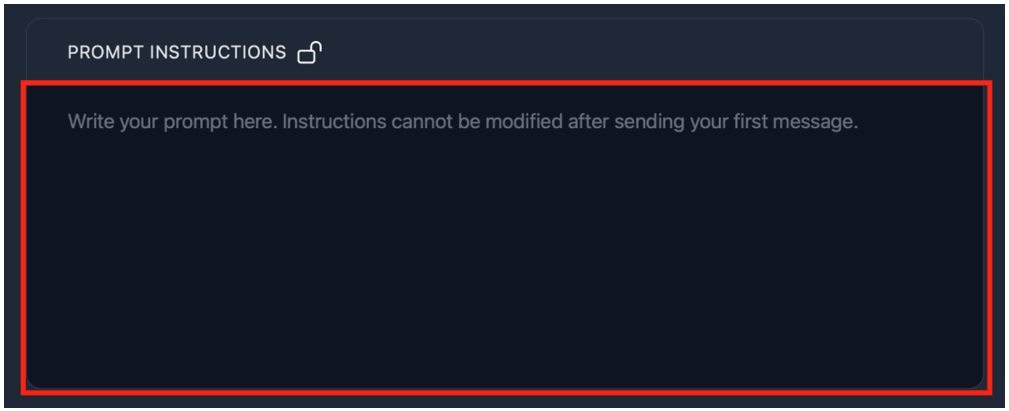


2. **Name the chat:** Utilize the pencil icon found at the top-left corner of the right pane to assign a name to the chat.

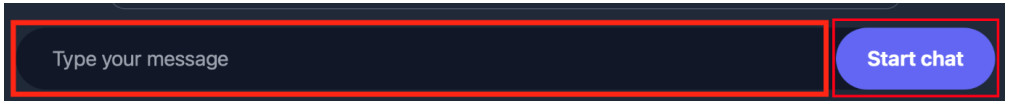


3. **Select the model:** Utilize the dropdown menu at the top-right corner of the pane to choose the **GPT-5 Nano** AI model for chat interactions. Here, by default, the model **Granite 3.2 8B (Reasoning)** is selected.

4. **Provide prompt guidance**: Access the Prompt Instructions field located in the upper right pane of the chat system to input specific instructions or contextual details regarding the required queries. Once the chat begins, the prompt will be fixed and cannot be altered thereafter.



5. **Enter your message:** Utilize the text box situated at the bottom of the page to input your question and interact with the chat system.



Once you have entered the **Type your message** field, proceed by clicking the 'Start Chat' button to initiate the conversation.

Task 2: Posing queries to ChatGPT

Once you have configured the AI Classroom, let's move forward and pose some questions to ChatGPT.

1. Please enter your query in the **Type your message** field to request guidance from ChatGPT regarding palindromes.

Question to chatgpt:

“What is a palindrome and how is it defined in terms of string or number sequences?”

▼ Click here to view the sample response generated

A palindrome is a sequence of characters that reads the same forward and backward. This applies to strings (any sequence of characters) or to numbers (the sequence of digits). Formally, a string s of length n is a palindrome if s[i] equals s[n-1-i] for all i in [0, n-1].

Palindromes in strings
Definition: A string is a palindrome if it reads identically from left to right and from right to left. By default, character case matters (e.g., 'Radar' is not the same as 'radar').

Palindromes in numbers
Definition: A number is palindromic if the sequence of its digits reads the same forward and backward (e.g., 121, 1331). Negative signs are typically not considered part of the digit sequence, so negative numbers are not palindromes under the common definition.

Common variations
For phrases or sentences, palindromes are often defined after normalizing the input: remove non-alphanumeric characters and convert to a consistent case, e.g., 'A man, a plan, a canal: Panama' becomes 'amanaplanacanalpanama' and is a palindrome.

Examples

- String:'radar' -> palindrome; 'hello' -> not a palindrome.
- Number: 12321 -> palindrome; 12345 -> not a palindrome.
- Phrase (normalized): 'Was it a car or a cat I saw?' -> palindrome when non-alphanumerics are ignored and case is normalized.

Algorithms
Simple approach: reverse the sequence and compare to the original. Efficient approach: use two pointers starting at the ends and move toward the center, comparing characters. For numbers, you can also test palindromicity without converting to a string by reversing half of the digits or by digit-by-digit comparison.

Edge cases

- Empty string and a single character are palindromes.
- For numbers, 0 is a palindrome.
- Negative numbers are not palindromes under the common definition unless the sign is ignored.

2. Please enter your query in the **Type your message** field to seek guidance from ChatGPT about the contrast between JavaScript and Java.

Question to chatgpt:

"What is the primary dissimilarity between JavaScript and Java?"

▼ Click here to view the sample response generated

Java and JavaScript are two distinct languages designed for different purposes. The primary dissimilarity is their typing and execution model: Java is statically typed and compiled to bytecode for the Java Virtual Machine (JVM), whereas JavaScript is dynamically typed and typically interpreted or Just-In-Time (JIT) compiled within a host environment (browsers or Node.js).

Details

- Typing: Java uses static (compile-time) typing; variables have declared types and type errors are caught at compile time. JavaScript uses dynamic (run-time) typing; types are resolved during execution and can change, which can lead to runtime type coercions.
- Compilation/Execution: Java compiles to bytecode and runs on the JVM. JavaScript is typically interpreted by a browser engine and often JIT-compiled at runtime for performance.
- Object model: Java is class-based with explicit class definitions and a clear inheritance structure. JavaScript is prototype-based and uses dynamic prototypes; ES6 introduced syntactic sugar like classes, but inheritance remains prototype-driven.
- Concurrency model: Java uses threads and a robust multi-threading model with shared memory. JavaScript uses a single-threaded event loop with asynchronous I/O (promises, async/await) and worker threads as an optional capability.
- Typical domains: Java is widely used for large-scale enterprise back-end systems, Android apps, and server-side services. JavaScript dominates web front-ends and is also used on the server (Node.js) and in various runtime environments.

3. Please enter your query in the **Type your message** field to request advice from ChatGPT regarding software development methodologies.

Question to chatgpt:

"What are the primary software development methodologies and their key differences?"

▼ Click here to view the sample response generated

Software development methodologies are structured approaches used to plan, execute, and deliver software projects. They guide how teams gather requirements, design, build, test, deploy, and maintain software, and they influence collaboration, risk management, and responsiveness to change.

Below are the primary methodologies, with their core ideas, typical practices, and where they tend to fit best.

Waterfall

- **Description**: A linear, sequential lifecycle where each phase must complete before the next begins.
- **Phases:** Requirements -> Design -> Implementation -> Verification -> Maintenance.
- **Characteristics:** heavy up-front planning, extensive documentation, limited flexibility for changes after a phase.
- **When to use:** well-understood requirements, minimal expected changes, regulatory environments.
- **Pros:** predictable schedules and budgets; clear milestones.
- **Cons:** late risk discovery and testing; high change cost; poor adaptability.

Agile (and its common frameworks)

- **Description:** An iterative, incremental approach that emphasizes customer collaboration and responding to change.
- **Core values (Agile Manifesto):** individuals and interactions, working software, customer collaboration, responding to change.
- **Common frameworks:**
  - **Scrum:** defined roles like Product Owner, Scrum Master, Development Team, time-boxed sprints, ceremonies, and artifacts.
  - **Kanban:** visual board, continuous flow, WIP limits, no required sprints.
  - **XP:** engineering practices like TDD, pair programming, continuous integration, refactoring.
- **When to use:** dynamic or uncertain requirements, need for rapid feedback, cross-functional teams.
- **Pros:** flexibility, faster delivery of valuable features, close customer involvement.
- **Cons:** requires disciplined teams, can lead to scope creep without strong product ownership.

Lean software development

- **Description:** Aims to minimize waste and maximize value by delivering what customers actually need.
- **Principles:** eliminate waste, amplify learning, decide as late as possible, deliver fast, empower teams, build integrity, optimize the whole.
- **Practices:** value stream mapping, Kanban-style flow, continuous improvement (kaizen).
- **When to use:** projects seeking efficiency, rapid iteration with minimal overhead, teams focusing on value.
- **Pros:** faster delivery with less waste; emphasis on customer value.

- **Cons:** can require cultural change; may be harder to scale without mature practices.

DevOps

- **Description:** A cultural and technical approach to unify software development and IT operations for faster, more reliable delivery.
- **Key practices:** continuous integration and delivery (CI/CD), infrastructure as code, automated testing, monitoring, and feedback loops.
- **Focus:** automation, collaboration, and measurement across the delivery pipeline.
- **When to use:** organizations needing rapid, reliable releases and strong operational feedback.
- **Pros:** shorter lead times, higher deployment reliability, better alignment between dev and operations.
- **Cons:** requires organizational and tooling investments; cultural barriers.

Spiral and V-Model

- **Spiral model:** Risk-driven, iterative approach that cycles through planning, risk analysis, engineering, and evaluation; emphasizes early risk mitigation and prototyping.
- **V-Model:** Emphasizes verification and validation with a corresponding testing phase for each development stage; common in safety-critical domains.
- **When to use:** Spiral for high-risk projects; V-Model for regulated or safety-critical systems where rigorous validation is required.
- **Pros/Cons:** Spiral can manage risk well but may be complex; V-Model provides strong testing discipline but can be rigid.

RAD and Prototyping / Feature-Driven Development (FDD)

- **RAD/Prototyping:** Emphasizes quick user feedback via rapid prototyping to refine requirements and design.
- **FDD:** Feature-Driven Development focuses on building by features guided by domain object modeling and a feature list.
- **When to use:** short time-to-market, when user feedback is critical; domains with clear feature sets.
- **Pros:** fast feedback loops; clear executable increments.
- **Cons:** can lead to scope drift if not controlled; requires good modeling and feature planning.

Choosing a methodology (practical guidance)

- **Consider requirements stability:** high stability favors Waterfall; high uncertainty favors Agile/Lean.
- **Regulatory and compliance needs:** may benefit from V-Model or Waterfall with strong documentation.
- **Team and culture:** cross-functional, collaborative teams fit Agile/DevOps; siloed teams may struggle.
- **Delivery goals:** need for rapid iterations? Agile/Lean; need rigorous testing and traceability? V-Model or Spiral.
- **Organization and tooling:** mature DevOps tooling supports CI/CD and automation; consider the cultural shift when adopting DevOps.

Key differences at a glance

- **Structure:** Waterfall is linear; Agile is iterative; Lean/DevOps emphasize flow and continuous improvement; Spiral/V-Model emphasize risk management and verification.
- **Requirements:** Waterfall assumes stable requirements; Agile/Lean tolerate changing requirements; Spiral manages risk with evolving requirements.

- **Planning horizon:** Waterfall long upfront planning; Agile short sprints and continuous planning; DevOps emphasizes ongoing delivery.
- **Customer involvement:** Waterfall moderate-to-low post-planning; Agile and Lean involve customers continuously; DevOps focuses on operations feedback as well.
- **Documentation:** Waterfall emphasizes documentation; Agile emphasizes working software and lightweight docs; DevOps favors automated configuration and runbooks.
- **Risk handling:** Waterfall mitigates risk late; Spiral explicitly manages risk; Agile/Lean manage risk via incremental delivery and feedback; DevOps reduces risk with automation.
- **Suitable contexts:** Waterfall for well-understood, stable domains; Agile/Lean for dynamic, customer-driven products; Spiral for high-risk; V-Model in safety-critical contexts; RAD for rapid prototyping.

Summary

Throughout this lab, you've delved into the potential of generative AI models for text generation. You've familiarized yourself with IBM's Generative AI classroom chatbot system, driven by multiple language models. You've also gained knowledge on creating desired text outputs using prompts and explored the text generation capabilities inherent in language models.

Congratulations!

You've completed the practical session aimed at acquainting yourself with the GenAi classroom setup and practicing text generation through generative AI.

Author(s)

Pallavi Rai