The AI-Enabled Instructor:
Three Methods for Enhancing Political Science Learning

Sean Peters[*†]

January 23, 2025

The explosion of artificial intelligence tools in recent years has created a rich environment for scholars, both in improved research and in the creation of novel methodologies. Yet faculty can leverage these tools in their capacities as instructors as well, particularly in conceptually abstract fields like the social sciences. I present three AI-enabled methods for enhanced instructional ability, each targeting a different component of the learning and instructional process, and unified under a framework known as ClassFactory. ClassFactory aims to assist instructors in three key areas of lesson preparation, course alignment, and student assessment. ClassFactory includes BeamerBot: an AI-powered lesson generator yielding a set of lecture slides in LaTeX; ConceptWeb: a module leveraging AI to perform relation extraction and community detection methods to generate interactive knowledge graphs; and QuizMaker: an automated static or interactive quiz generator drawing directly from course readings. Taken together, these methods have the potential to significantly reduce instructor administrative workload while simultaneously enhancing student learning.

Introduction

Full-time academic faculty face a daunting challenge in time management, balancing teaching duties with service, research, advising, professional development, and administrative responsibilities. Over time, faculty hours have tended to increase, with more time spent on both teaching and research (Milem, Berger, and Dey 2000). Combining a standard teaching load with the additional career incentives associated with academic promotion (Link, Swann, and Bozeman 2008) can lead to struggles with work-life balance resulting in conflict with one's family life (Dahm et al. 2015; Minnotte 2021), often with differential effects by gender (Winslow 2010; Allen et al. 2023). In this context, the need for tools to improve faculty's ability to balance the varied demands on their time becomes clear. AI-powered methods present a useful approach to improving efficiency and reducing faculty's time allocation burden. The ClassFactory package leverages these tools to create a structured workflow that assists instructors through a blend of automation and AI-driven solutions for class preparation. By automating the creation of slides, knowledge graphs, and assessments, instructors can make more efficient use of their limited instructional time to focus their efforts toward higher-order tasks, such as refining course design or engaging with students more deeply.

[*]Assistant Professor of Political Science, United States Air Force Academy

[†]The views expressed are those of the author and do not reflect the official guidance or position of the United States Government, the Department of Defense, the United States Air Force or the United States Space Force.

AI As An Efficiency Booster

The advent of large language models (LLMs) has led to an explosion of research and tools built off of their generative capacities. Innovations like the transformer architecture (Vaswani et al. 2017) led to the creation of language models like BERT (Devlin et al. 2019) and of generative models like GPT (Radford et al. 2018). After being trained using a mix of next sentence prediction and masked language modeling (Vaswani et al. 2017), these foundation models have been shown to excel at processing unstructured text data and performing tasks in a few-shot or even zero-shot manner (Brown et al. 2020), simply by predicting the next most probable word or sentence. Moreover by adjusting the context provided by the user, these models demonstrate a capacity for "in-context" learning without additional fine-tuning or model weight updates (Radford et al. 2019). If we combine LLMs' natural language understanding capabilities with clear and specific few-shot or zero-shot prompts, we can leverage these models to automate a variety of otherwise time-intensive tasks.

While instructors cannot automate away their own reading and intellectual class preparation, they may expend a significant amount of time and effort preparing class materials or assessments. Tasks such as formatting slides and quizzes are often repetitive, yet they can still take up significant amounts of time. The tasks literature in economics would likely classify these functions as "routine" (Acemoglu and Autor 2011) in that they do not require higher-order skills; yet until recently they were not subject to a rule-based or computer-driven solution. The advent of large language models and their ability to understand raw natural language, expands the breadth of routine tasks which can be automated. While the tasks literature focuses on how routine tasks may be subject to outsourcing to other labor markets, the growth in power and accessibility of large language models allows for such routine tasks to be delegated instead to AI-powered systems.

Moreover, ClassFactory presents a solution to a problem inherent in typical LLM chat interfaces. By virtue of their probabilistic generation modalities, large language models do not provide deterministic answers to a given prompt. Simply uploading a reading to a chat model carries an inherent risk of hallucination and poor response quality or structure. For example, a simple query to a LLM chat interface to generate a set of lecture slides might generate a LaTeX document, but (1) it might not compile correctly due to syntactic errors; (2) it might hallucinate and fail to provide a response that is faithful to the context provided; and (3) it might simply fail to adequately follow prompt instructions. If repeated for multiple iterations of lesson material generation, a user might prompt the model in different ways at different times, introducing even more variability into model output. To illustrate this point, the appendix shows a simulation of the above scenario, in which a user sends a query to OpenAI's `GPT-4o`, a model more powerful than was used in testing for ClassFactory, providing lesson objectives and readings; and asking the model to create a Beamer slideshow (akin to BeamerBot's functionality, to be discussed below). The slides created were of generally acceptable quality; but the LaTeX would not compile due to shortfalls in the LaTeX script preamble (see Appendix B). ClassFactory, on the other hand, prompts the model in a consistent manner, ensures consistent structured

output, automatically checks for correct LaTeX compilation, and validates both quality of response and fidelity to provided context. Any failures of these automated checks will prompt an automatic retry to ensure consistent model output and quality. Importantly, ClassFactory's structured generation process leads to more successful generation even when using a weaker model (most testing was accomplished using `gpt-4o-mini`). The structured inputs and outputs, combined with automatic validation and retry logic, provides a significant benefit over ad hoc LLM use for otherwise tedious tasks.

The outline of this paper is as follows. First I describe the structural features and components of the ClassFactory package. Next I provide example implementation for each of its submodules. The paper concludes with a discussion of this package's limitations and prospects for enhancement. The ultimate goal of this project is to provide instructors with an easy to implement solution to the routine tasks of class preparation and assessment.

ClassFactory Structure and Components

The ClassFactory package consists of three submodules, each with its own distinct function: BeamerBot for automated LaTeX slide generation, ConceptWeb for concept map (a form of knowledge graph) generation, and QuizMaker, for assessment generation and hosting.[1] The ClassFactory object creates a unified interface to each of the three different modules using a "concrete factory" design pattern implemented in Python (Gamma 1995). After creating the ClassFactory object, the user can call any of the three modules from the ClassFactory instance, all of which will inherit the basic data provided to the factory object.[2] The core functionality across factory modules follows three steps:

1. **Document Parsing and Loading**: Ingest readings (DOCX, PDF, or TXT) and syllabus data, with optional OCR for non-readable PDFs.
2. **Module Instantiation**: Use the factory class to call the desired module. While the module may be called directly, each module inherits the document and directory data from the factory class, allowing the user to create multiple products from the same base factory.
3. **Model Prompting**: Custom-built prompts tailored for each submodule task (i.e., BeamerBot, ConceptWeb, QuizMaker), which are designed to be agnostic to specific LLMs.
4. **Response Validation**: Ensures accuracy and fidelity through LLM-based quality checks and feedback loops leveraging LLM-as-a-judge techniques (Gu et al. 2024).

First, the user provides the module a directory within which lesson readings are located, as well as a path to the course syllabus. The factory object will load all readings within subdirectories associated with the provided lesson number or range.[3] The factory will use a rule-based search to

1. Documentation can be found at https://speters9.github.io/ClassFactory/class_factory.html

2. While this package requires the user to have some facility with Python, the factory implementation abstracts away most coding requirements such that the user can simply instantiate an LLm, the factory and module, and then call the module's primary function. Consequently user familiarity with Python can be quite basic and still yield good results.

3. It accepts DOCX, PDF, or TXT files. Optional OCR capabilities exist if the user has PDF files that are not readable, however this functionality requires additional dependencies

parse the provided syllabus to determine the lesson-specific objectives (if any) and readings for the provided lesson number. The user must also provide a LLM object to be invoked.[4]

Once the factory has loaded the requisite files, the user will create one of the three submodules: BeamerBot (for automated slide generation), ConceptWeb (for knowledge graph generation), or QuizMaker (for quiz generation and hosting). Each module has built-in prompts that have been engineered for their respective purposes. The default prompt is formatted to include the readings, lesson objectives,[5] and course name, as well as additional guidance. Next, the model is invoked and its response is parsed to ensure correct formatting. In all cases, the ClassFactory modules rely on the now significant context windows that state of the art models offer. For example, OpenAI's GPT-4o has a context window of 128,000 tokens (roughly 96,000 words) (OpenAI 2024b, 2024a). If we assume roughly 300 words per page of text, this equates to about 320 pages of text the model can accept as context.[6] In any but the most reading-intensive course, this context window should be more than sufficient to include all readings for an undergraduate reading load on a given lesson within one prompt. Importantly, each module provides an example formatted output to the model to ensure alignment with desired outputs and consistent quality via in-context learning.

A key feature of ClassFactory is its built-in automatic response validation using LLM-as-a-Judge (Gu et al. 2024). This validation mechanism enhances reliability, making ClassFactory a robust tool for educators who may otherwise hesitate to adopt AI due to concerns about accuracy or hallucination. After response parsing, a separate validator class is queried, requesting the LLM examine the user prompt and original response to ensure accuracy and fidelity to the provided context. The validator provides a quality score of the LLM's initial response. If validation fails (i.e. the score falls below a user-defined threshold), it will return additional guidance to adjust the prompt provided to the LLM and then automatically retry the task. While this does not completely solve the problem of model hallucination (Huang et al. 2024; Maynez et al. 2020), it provides a backstop to ensure a level of fidelity to the provided context.[7] Once the model's response has been validated, the respective module will process the response according to its unique functionality. We turn next to each of the three ClassFactory submodules.

BeamerBot: Automated Slide Generation

A particularly time-consuming task–especially for new instructors–is the generation of new lesson content. While some may borrow products from other instructors, the initial process remains time-consuming as each instructor adjusts the content to align with their respective pedagogical id-

4. Results and most testing was accomplished using OpenAI's `gpt-4o-mini` but the factory will accept any LLM if passed via its respective `Langchain` library wrapper. For LLM's run locally, LLaMA 3.1 was also used with some success, although the quiz questions generated were of a lower quality.

5. If the syllabus does not contain lesson-specific learning objectives, the modules will simply prompt the model without them. Lesson-specific objectives help align the model better to a given lesson, but it functions satisfactorily using only the syllabus readings and other lesson information from the syllabus as context.

6. It must be noted that the quality of recall within these context windows may vary, as pointed out by Liu et al. (2023).

7. The user should note that this requires a second LLM call, which might be of concern to cost-conscious users.

iosyncrasies. BeamerBot seeks to resolve this by providing an LLM with an example slideshow[8] as well as the lesson readings and lesson objectives to create a baseline slide deck from which the instructor can build their full lecture.

BeamerBot allows for the user to specify their own prompt for the model; the default prompt instructs the model to provide a structured slideshow including prior and subsequent lesson information, creation of an in-class exercise, slides based on the reading content (as structured by the lesson objectives, if provided), and key takeaways for the students. Users may easily adjust the prompt to change the number of content slides in order to align with the desired lecture length. The resulting slideshow creates a framework for a full lesson that can then be built out by the instructor. It must be noted here that BeamerBot slideshows are to be considered a starting point, and not a final solution for the lesson itself. Nevertheless BeamerBot can significantly lessen the instructor preparation load by automating repetitive tasks while creating a consistent slide format and flow for instructor and student continuity.

ConceptWeb: Automated Knowledge Graph Generation

A common struggle for students is understanding how the concepts they are learning fit together as part of a more coherent narrative. Faculty aim to guide their students beyond simple memorization to an actual understanding of the connections between the wide variety of ideas covered in class, and organizing them into a gestalt whole. Connecting and organizing disparate facts into a coherent body of knowledge represents a significant challenge for students, as such learning sits at a relatively advanced level in Bloom's taxonomy of learning (Anderson et al. 2000). The ConceptWeb module seeks to surmount this difficulty by creating interactive visual representations of the knowledge contained in course readings. These representations can assist students in making the leap from factual understanding to conceptual application, while also helping instructors assess their course design, ensuring the desired themes and concepts are sufficiently covered.

At the core of ConceptWeb is a knowledge graph, a relational data structure consisting of sets of triplets taking the form, in this case, of [concept - relationship - concept]. These triplets are arranged into a graph structure, where each concept becomes a node, and each relationship between concepts becomes an edge (Nickel et al. 2016). ConceptWeb automates the construction of this graph by processing course readings in a multi-step pipeline.

First, ConceptWeb feeds the LLM a given course text and instructs the model to summarize the text in a structured output of 250-350 words, balancing substantive detail with output brevity. The summary, along with lesson objectives (if provided), is passed to the model a second time to extract key concepts and relationships (e.g. ["X", "causes", "Y"]) from the summary in light of the lesson's objectives. These extracted triplets are validated and stored for further processing.

8. Note: this module is designed for LaTeX Beamer slides. It does not yet have functionality for PowerPoint or other slides, although there are plans to incorporate this.

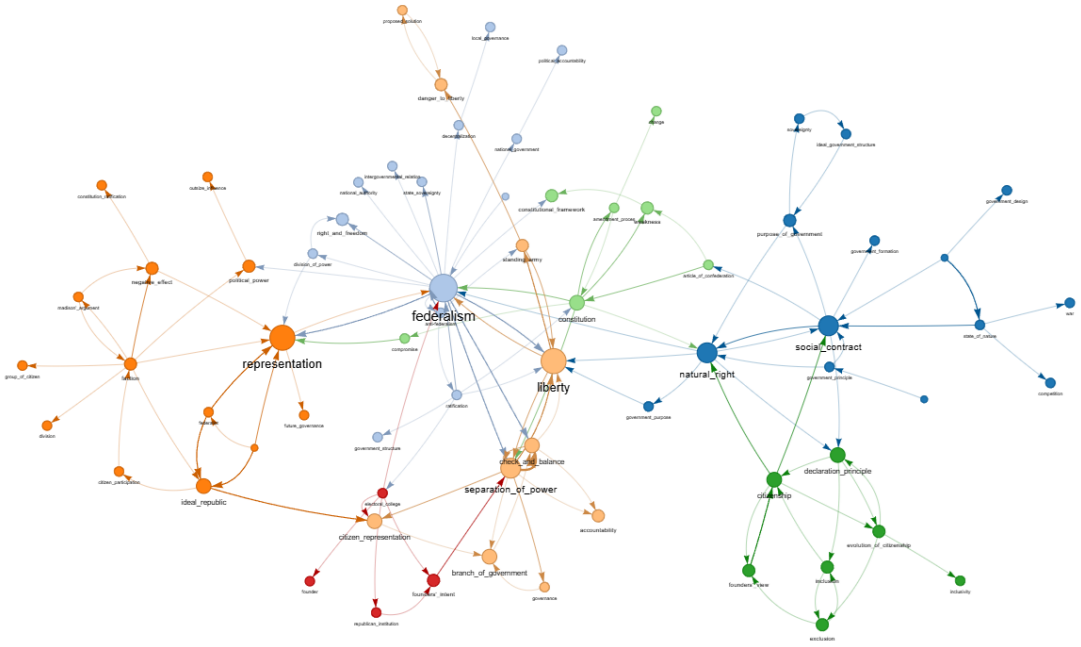

Figure 1: Example Concept Map

Due to the probabilistic nature of LLMs and the inherent variability in their responses, it can sometimes be the case that the model may return similar concepts in slightly different ways, resulting in a knowledge graph with conceptually identical but textually distinct nodes. To handle potential duplication in concept representation (e.g., "Congress" vs. "Congressional"), ConceptWeb employs an entity resolution process. Each concept is converted into a vector embedding using DistilBERT (Sanh et al. 2020), a lightweight version of the BERT model (Vaswani et al. 2017). By comparing embeddings via cosine similarity, ConceptWeb identifies and merges similar concepts into a unified representation. This ensures that the prominence and recurrence of distinct conceptual representations are displayed accurately in the resulting knowledge graph.

After entity resolution is complete, the concepts and their relationships are then consolidated into a knowledge graph. The user may choose to generate either a directed or undirected graph; but for a basic understanding of relationships between concepts (i.e. a general understanding of how and which concepts have any relationship to each other at all), ConceptWeb's default output is an undirected graph (See Figure 3 for an undirected concept map generated from readings of Locke, Montesquieu, and Hobbes). Next, for deeper analysis, ConceptWeb applies a community detection algorithm (Traag, Waltman, and Van Eck 2019), grouping related concepts into clusters that in practice tend to correspond to distinct syllabus topics or lessons. The user may zoom or hover over nodes and edges to see more information (as well as the labels for smaller nodes); they can also rearrange the graph into a hierarchical rather than clustered structure.

Finally, ConceptWeb creates and saves an interactive visual depiction of the graph using the python library `plotly`. ConceptWeb's graph visualization scales the node size by concept recurrence across relationship triplets, and the edge size by the frequency of relationships between distinct con-

cepts. Node and edge colors correspond to their discovered community. The result is an unsupervised concept-based visual interactive graph, drawn directly from course materials and illustrating the key concepts as well as the strength of their relationships with other course concepts. Figure 1 shows a concept map spanning ten lessons' worth of content, using a directed graph.

QuizMaker: Automated Quiz Generation and Hosting

The final module in ClassFactory is QuizMaker. This module uses a similar approach as ConceptWeb, looping through lessons in a user-provided lesson range, and then individual readings, validating LLM response accuracy along the way. However instead of extracting key concepts and relationships, the LLM is instructed to generate three different types of questions from a given reading: multiple choice, true/false, and fill-in-the blank. Each of these is designed to be returned via a standardized JSON format to ensure compatibility with future processing and export tools.

To ensure originality and prevent overlap with previously generated content or upcoming assessments, QuizMaker allows the user to pass a list of questions which the LLM will avoid generating. It also performs cosine similarity checks between LLM-generated quiz questions and any user-provided questions to discard any overly similar questions. This ensures quality output that does not overlap with either previously generated or upcoming assessment questions. The generated quiz questions can be exported to an Excel spreadsheet or PowerPoint presentation for use in class.

Finally, QuizMaker allows users to present generated quizzes interactively with students. Using the library `Gradio`, QuizMaker generates a customizable quiz interface and QR code for easy student access. As students take the quiz, their anonymized results are saved to the user's machine. Once students have finished taking the quiz, the user can automatically analyze quiz results to identify knowledge gaps, share with students, or adjust their instruction in future lessons.

Using ClassFactory

This section outlines the basic functionality of the package. Figure 2 demonstrates the workflow for all factory modules. The package simplifies the generation of teaching materials by managing class readings, parsing syllabus objectives, and leveraging the user-defined large language model (LLMs) within the ClassFactory class itself. Each module builds on this baseline functionality for its respective tasks to avoid duplicated effort.

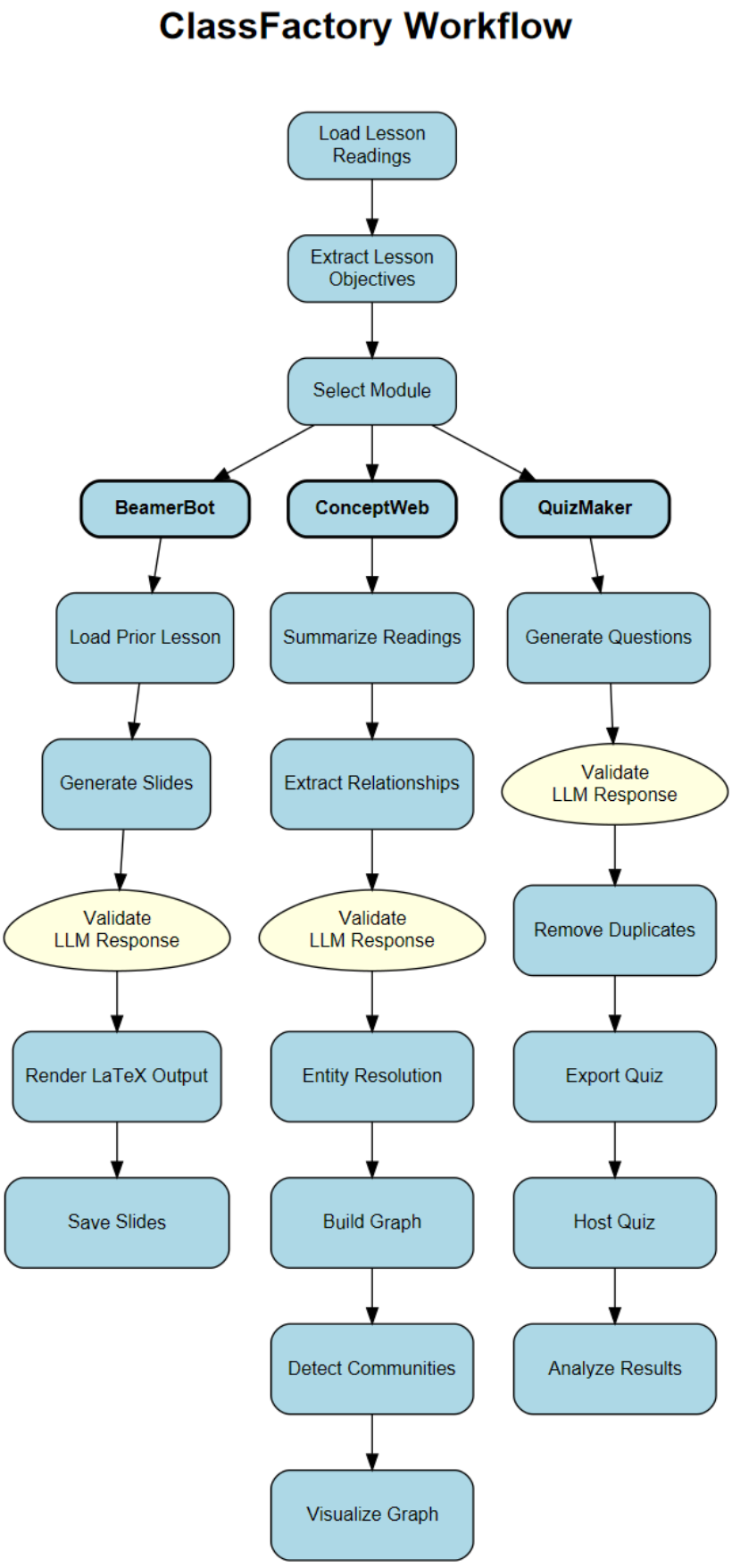

Figure 2: ClassFactory Workflow

Workflow Overview and Factory Initialization

All products draw their information directly from course readings. To access these readings, ClassFactory assumes a specific directory structure. Specifically, it assumes a parent directory (`'readingsDir'` below), with subdirectories identified by their corresponding lesson number. For example:

Code Chunk 1: Required Reading Directory Structure

```
readingsDir/                <- Directory of Directories
    |-- L1/                 <- All readings for Lesson 1
    |-- L2/                 <- All readings for Lesson 2
    ... etc
```

After providing a lesson number or range, ClassFactory will scan the provided readings directory for all lessons within that range and process those readings (DOCX, PDF, or TXT). The base installation of ClassFactory assumes all PDF files contain readable text; an optional installation will provide OCR functionality, but this requires installation of external dependencies.[9]. For this demonstration we will be drawing readings from two lessons of a course on American government. The readings include selections from Hobbes, Locke, and Montesquieu that all contain readable text.

To guide LLM response, ClassFactory takes as user input a course syllabus (PDF or DOCX). The factory object will parse the syllabus in search of the information for the provided lesson at runtime, using a rule-based approach to identify keywords like "Lesson," "Week," or "Lecture" followed by integers. Alternatively, the user may provide objectives for each lesson to guide the LLM generation if the syllabus is unavailable or if the syllabus does not contain explicit objectives for each lesson. Lesson-specific learning objectives help align the LLM's outputs with instructor intent, but ClassFactory will function satisfactorily if no lesson-specific learning objectives are provided.

Next, we instantiate an LLM object that will serve as the workhorse for each of the ClassFactory modules. To interface with the LLM we rely on the `Langchain` library. If you plan to use an API-based model like OpenAI's GPT-4 or Anthropic's Claude, you must also provide the model with your API key.[10] The example that follows will be running the example code in `'run_classfactory.py'` in the ClassFactory repository on GitHub.[11] We begin by creating the LLM and factory objects:

Code Chunk 2: Initial Setup

```
# We assume required imports have been completed and directories are
    structured appropriately. See run_classfactory.py in repository for
    necessary imports

llm = ChatOpenAI(
    model="gpt-4o-mini",  # gpt-4o-mini has a good balance of quality and cost
    temperature=0.1,      # control model creativity; 0 is least creative
    api_key=OPENAI_KEY)
```

9. External installations include Tesseract OCR, Poppler, and spaCy

10. For secure credential handling, it is recommended the user manage this information using environment variables rather than hard coding API keys.

11. https://github.com/speters9/ClassFactory

```
7
8  factory = ClassFactory(lesson_no=3,
9                         syllabus_path=syllabus_path,
10                        reading_dir=readingDir,
11                        llm=llm,
12                        course_name="American Government",
13                        lesson_range=range(2, 4), # This will look for
                              directories within the readings directory titled "
                              L2", "L3", etc.
14                        verbose=False)
```

For this example we use OpenAI's `gpt-4o-mini`, although all modules have had good success with Anthropic's Claude and Google's Gemini.[12] We then pass the defined directories as well as the llm to the ClassFactory object, as demonstrated in Code Chunk 2. Upon initialization it will create a lesson loader object that will manage all file processing for the subsequently created modules.

We pass a course name to the ClassFactory object because this is passed as context for the LLM later; creating a role for the model helps it generate more specific and relevant content (Sherman et al. 2024), and allows for more flexible content generation across a variety of instructional courses. When provided with a lesson range, the ConceptWeb and QuizMaker classes will load readings for every lesson in the range.[13]

All ClassFactory modules include a built-in process for validating LLM outputs–ensuring consistency in both content and format. During the generation process in any module the user may see validator warnings pop up, specifying the validator's concerns with the language model's response as well as a suggested update to the baseline prompt to fix the issue. This is normal, and provides the user with context on specific areas in which the LLM may be struggling to perform its task. Validation errors are most common when using ConceptWeb (due primarily to the large number of readings processed); however the module will automatically retry its query using the validator's suggested verbiage, so these warnings are mostly for user information.

Individual Module Creation and Use

After creating the factory class, creation and use of factory submodules is relatively straightforward. The user calls the `create_module()` method from the ClassFactory, specifying the module to create.[14] Code Chunk 3 shows the initialization and basic operation of BeamerBot.

BeamerBot Example Usage

Code Chunk 3: BeamerBot Initializiation

```
1
2  beamerbot = factory.create_module("BeamerBot",
```

12. The open source LLaMa model has had some success but was less reliable due to restricted model size stemming from system GPU limitations. Using the 70B or 405B versions of the LLaMa would likely yield higher quality results. Mistral also performed quite well but struggled with longer readings due to its smaller context window.

13. BeamerBot is designed to build a slide deck for one lesson only.

14. For BeamerBot the user must also pass the directory where prior lesson slides are stored. The naming convention on these slides should mirror those of the directory structure (eg "L1.tex," "L2.tex," etc.)

```
                                    verbose=False,
                                    slide_dir=slideDir)
                                     # slideDir is the directory where prior
                                        lesson slides are stored
slides = beamerbot.generate_slides()  # Create the LaTeX slide deck
beamerbot.save_slides(slides)
```

Note that the above example saves the generated slides. When initialized, each module creates its own directory under a broader "ClassFactoryOutput" directory in the project's root folder. The slides will be saved at `project_root_directory / ClassFactoryOutput / BeamerBot /` with a folder for the associated lesson number. Similar logic follows for outputs saved from the other modules.[15]

The user can also add specific requirements to BeamerBot as it creates slides. This is easily accomplished using the `specific_guidance` argument. Note in the below code chunk that guidance to the model is passed using Markdown format. Using Markdown has helped provide consistent model output and aligns with the formatting in the primary model prompts (see primary model prompts in the appendix):

Code Chunk 4: BeamerBot Slide Generation with Specific Guidance

```

specific_guidance = """
Add the below section to its own slide:
---
## Slide Name:
 - Point 1
 - Point 2
 - Point 3
"""

slides = beamerbot.generate_slides(specific_guidance=specific_guidance)
```

ConceptWeb Example Usage

The same functional flow applies across the other ClassFactory submodules: create the module, run the primary function, and save results. The factory method ensures a high-level interface with each of the submodules for easier user interaction. Code Chunk 5 below demonstrates the instantiation of the ConceptWeb module and generation of a concept map for the designated lesson range. Figure 3 shows the concept map generated using this code:[16]

Code Chunk 5: ConceptWeb Initializiation

```

builder = factory.create_module("ConceptWeb", verbose=False)
builder.build_concept_map(directed=False, dark_mode=False)     # saves concept
     map and a wordcloud of concepts
```

15. The user can provide their own desired output directory as well.

16. As a reminder, LLMs are probabilistic models. We cannot guarantee strict reproducibility of a specific concept map from a given reading. However setting a very low model temperature (e.g. 0 or 0.1) will help ensure consistent–albeit still not deterministic–responses.

If the returned concept map contains too many redundant concepts (or too few concepts), the user might consider adjusting the `concept_similarity_threshold` argument, which defaults to 0.995.[17] For a helpful rule of thumb, the user might consider adjusting the threshold by increments of 1.5×10^{-3} to fit their desired outcome. The result, as shown in Figure 1 and Figure 3, is a visual interface allowing students to directly interact with the primary concepts found in their course readings.

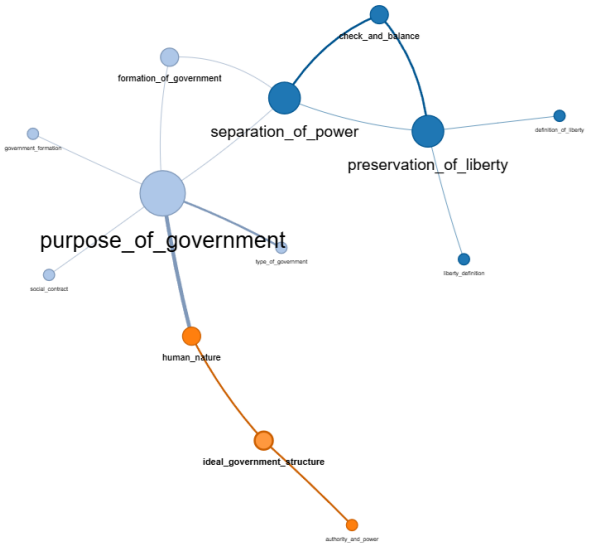

Figure 3: Example Concept Map

QuizMaker Example Usage

The QuizMaker module unifies assessment generation, hosting, and analysis into one module. Its instantiation follows a similar logic to the first two modules. If desired, the user can provide a directory in which all quizzes are stored (either prior or upcoming assessments); these questions will be provided to the model as a list of questions not to generate.[18] It then follows the same pattern as the above modules:

QuizMaker allows the user to specify how difficult a generated question should be on a scale of 1-10. Scores 1-3 represent simple recall-type questions; difficulty scores of 4-7 should require students to understand the relationship between concepts; and scores of 8-10 require application, analysis, and synthesis of ideas from the reading. See Code Chunk 11 in the appendix for details provided to the LLM in its prompt. Note also the `flag_threshold` argument provided to QuizMaker above: this allows the user to specify the level of acceptable question similarity: a lower threshold indicates a higher level of required uniqueness. Table 1 shows a sampling of questions generated using the above methods. Questions were generated from the three course texts: Locke, Montesquieu, and Hobbes; and represent the three types of questions that QuizMaker generates (true/false, multiple choice, fill-in-the-blank).

Code Chunk 6: QuizMaker Initializiation

```
quizDir = wd / "data/quizzes/"
quizmaker = factory.create_module("QuizMaker",
                                  lesson_range=range(1, 4),
                                  prior_quiz_path=quizDir,
                                  verbose=False)

quiz = quizmaker.make_a_quiz(flag_threshold=0.7, difficulty_level=9)
```

17. Given that the word embeddings are all in the same relative multidimensional space because they all pertain to the same topics, we should expect a high level of baseline similarity. Consequently a high bar is required to ensure the entity resolution process does not collapse all concepts into one.

18. QuizMaker expects Excel files of the following columnar structure: ['type', 'question', 'A)', 'B)', 'C)', 'D)', 'correct_answer']

If satisfied with the generated questions, QuizMaker allows the user to save generated quiz materials to a PowerPoint presentation for student review or in-class presentation. The quiz can also be saved to a simple Excel spreadsheet. Users can save generated questions using the `save_quiz()` or `save_quiz_to_ppt()` methods.

Once the quiz has been generated, the user can distribute it for student assessment. QuizMaker's interactive hosting functionality is a feature that particularly distinguishes it from the other ClassFactory modules. QuizMaker uses the library `Gradio` to launch and host an active quiz link for students to use. To access the link, QuizMaker generates a QR code for dissemination upon quiz launch. Quiz results will be saved directly to the user's machine for further analysis. An important caveat is that the user, when launching this functionality, must ensure continued network connectivity, as the quiz itself is being hosted from the user's machine; loss of internet access will require the user to re-launch the quiz. When launching the quiz, QuizMaker will accept as an argument either a path to a previously saved quiz (in Excel format), or the recently generated quiz object.

Question Type	**Example Question and Answer**
True/False	*Question*: "Hobbes argued that individuals should retain the right to overthrow their sovereign if they are dissatisfied." *Answer*: False
Fill-in-the-Blank	*Question*: "Montesquieu's idea of _____ is critical to maintaining checks and balances within government." *A)* Democracy *B)* Liberty *C)* Separation of Powers *D)* Federalism *Answer*: Separation of Powers
Multiple Choice	*Question*: "Which of the following best describes John Locke's view on human nature?" *A)* Humans are inherently selfish and violent *B)* Humans are rational and capable of self-governance *C)* Humans are dependent on the state for their morality *D)* Humans are naturally inclined to form oligarchies *Answer*: B

Table 1: Sample Questions Generated by QuizMaker

By default, QuizMaker generates six questions from each reading (two of each type), resulting in a relatively large bank of questions. To account for this, QuizMaker randomly samples from generated quiz questions before launching the interactive quiz. The user can adjust the sample size and random seed as desired. Code Chunk 7 provides an example implementation of launching the interactive quiz of ten questions and then analyzing the results.

After students have completed the assessment, QuizMaker can automatically analyze the results that have been saved on the user's machine, outputting both overall summary statistics and an interactive HTML document showing overall statistics as well as commonly missed questions. This consolidated format can be helpful for a student's learning as well as for the instructor to easily identify common knowledge gaps across students. Figure 4 shows an example quiz assessment report

from responses to the generated questions.

Code Chunk 7: Launch and Analyze Quiz

```
# On quiz launch, a QR code will be saved in QuizMaker output directory.
# The module will provide the user with a path to the QR code.
quizmaker.launch_interactive_quiz(quiz_data=quiz,
                                  sample_size=10,
                                  save_results=True,
                                  seed=80920,
                                  qr_name="quiz_review")

quizmaker.assess_quiz_results()
# If analyzing a different quiz, simply provide the directory containing saved
    quizzes using the 'results_dir' argument
```

By following the above steps, QuizMaker operates as an end-to-end solution for creation, hosting, and analysis of student assessments. The questions are generated directly from course texts, and aligned with course objectives. Interactive hosting and logging of results allows instructors to draft and launch a novel quiz within minutes, with the ability to analyze and present the results to students upon completion. While modern quiz software can easily handle real-time hosting and analysis, it still requires time-consuming manual drafting and uploading of questions to the hosting service. QuizMaker combines all the functions–generation, hosting, and analysis–under one streamlined module.

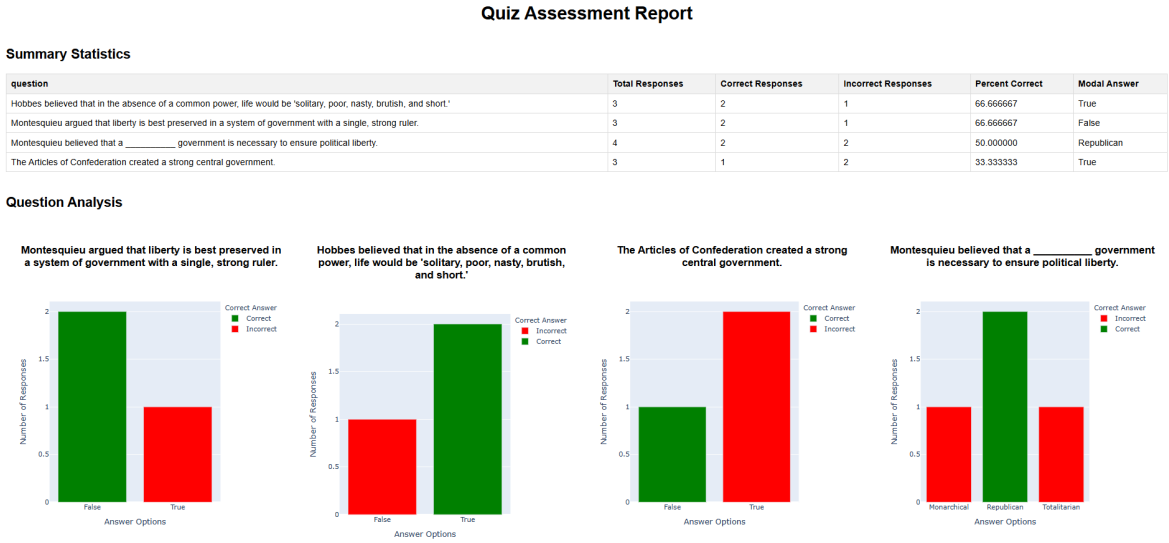

Figure 4: Sample Quiz Analysis

Conclusion

Collectively, the ClassFactory package presents a three-part solution to reduce instructor workload while enhancing student experience. From streamlining class preparation (BeamerBot), facilitating course design and instruction (ConceptWeb), and automating assessment (QuizMaker), ClassFactory demonstrates the potential of AI to enhance both the effectiveness and efficiency of instructors' efforts.

However, as with any AI-generated product, each module's output is subject to a level of

variability and occasional hallucination. Each module seeks to mitigate these risks by leveraging the large context offered by state-of-the-art LLMs and providing the readings and lesson objectives directly into the context (Li et al. 2024), as well as by validating output correctness and fidelity to course texts by using LLM-as-a-Judge (Gu et al. 2024), but instructors must still ensure quality of the returned product. The user should exercise principles of responsible use and *always validate model output before passing to students*. This point cannot be emphasized enough. The above modules should not serve as a replacement for instructor effort; rather they augment it. BeamerBot can create an excellent starting point for a lecture, for example, but each instructor must make the lecture their own. Similarly the generated QuizMaker questions should be analyzed for correctness and alignment with course objectives.

The demands on a university professor's time are manifold. By leveraging AI-powered techniques, instructors can direct their efforts in more efficient ways, allowing them to focus on more knowledge-intensive tasks such as individual student interactions, course design, etc. This will lead to enhanced class preparation and execution, while simultaneously improving the student experience. An AI-enabled instructor utilizing the above techniques can make the classroom more dynamic, effective, and engaging.

References

Acemoglu, Daron, and David Autor. 2011. "Skills, Tasks and Technologies: Implications for Employment and Earnings." In *Handbook of Labor Economics,* 4:1043–1171. Elsevier. ISBN: 978-0-444-53452-1, accessed November 23, 2024. https://doi.org/10.1016/S0169-7218(11)02410-5. https://linkinghub.elsevier.com/retrieve/pii/S0169721811024105.

Allen, Tammy D., Michelle Hughes Miller, Kimberly A. French, Eunsook Kim, and Grisselle Centeno. 2023. "Faculty Time Expenditure Across Research, Teaching, and Service: Do Gender Differences Persist?" *Occupational Health Science* 7, no. 4 (December): 805–818. ISSN: 2367-0134, 2367-0142, accessed December 18, 2024. https://doi.org/10.1007/s41542-023-00156-w. https://link.springer.com/10.1007/s41542-023-00156-w.

Anderson, Lorin, David Krathwohl, Peter Airasian, Kathleen Cruikshank, Richard Mayer, Paul Pintrich, James Raths, and Merlin Wittrock. 2000. *Taxonomy for Learning, Teaching, and Assessing, A: A Revision of Bloom's Taxonomy of Educational Objectives, Abridged Edition.* 1st edition. New York: Pearson, December 19, 2000. ISBN: 978-0-8013-1903-7.

Brown, Tom B., Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, et al. 2020. *Language Models are Few-Shot Learners,* arXiv:2005.14165, July 22, 2020. Accessed November 8, 2024. arXiv: 2005.14165. http://arxiv.org/abs/2005.14165.

Dahm, Patricia C., Theresa M. Glomb, Colleen Flaherty Manchester, and Sophie Leroy. 2015. "Work–family conflict and self-discrepant time allocation at work." *Journal of Applied Psychology* 100, no. 3 (May): 767–792. ISSN: 1939-1854, 0021-9010, accessed December 18, 2024. https://doi.org/10.1037/a0038542. https://doi.apa.org/doi/10.1037/a0038542.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,* arXiv:1810.04805, May 24, 2019. Accessed April 29, 2023. https://doi.org/10.48550/arXiv.1810.04805. arXiv: 1810.04805[cs]. http://arxiv.org/abs/1810.04805.

Gamma, Erich, ed. 1995. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley professional computing series. Reading, Mass: Addison-Wesley. ISBN: 978-0-201-63361-0.

Gu, Jiawei, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, et al. 2024. *A Survey on LLM-as-a-Judge,* arXiv:2411.15594, December 16, 2024. Accessed December 17, 2024. https://doi.org/10.48550/arXiv.2411.15594. arXiv: 2411.15594[cs]. http://arxiv.org/abs/2411.15594.

Huang, Lei, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, et al. 2024. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,* arXiv:2311.05232, November 19, 2024. Accessed November 23, 2024. https://doi.org/10.48550/arXiv.2311.05232. arXiv: 2311.05232. http://arxiv.org/abs/2311.05232.

Li, Xinze, Yixin Cao, Yubo Ma, and Aixin Sun. 2024. *Long Context vs. RAG for LLMs: An Evaluation and Revisits,* arXiv:2501.01880, December 27, 2024. Accessed January 18, 2025. https://doi.org/10.48550/arXiv.2501.01880. arXiv: 2501.01880[cs]. http://arxiv.org/abs/2501.01880.

Link, Albert N., Christopher A. Swann, and Barry Bozeman. 2008. "A time allocation study of university faculty." *Economics of Education Review* 27, no. 4 (August): 363–374. ISSN: 02727757, accessed December 18, 2024. https://doi.org/10.1016/j.econedurev.2007.04.002. https://linkinghub.elsevier.com/retrieve/pii/S0272775707000623.

Liu, Nelson F, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. "Lost in the Middle: How Language Models Use Long Contexts," arXiv: {arXiv}:2307.03172.

Maynez, Joshua, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. "On Faithfulness and Factuality in Abstractive Summarization." In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,* edited by Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, 1906–1919. ACL 2020. Online: Association for Computational Linguistics, July. Accessed November 23, 2024. https://doi.org/10.18653/v1/2020.acl-main.173. https://aclanthology.org/2020.acl-main.173.

Milem, Jeffrey F., Joseph B. Berger, and Eric L. Dey. 2000. "Faculty Time Allocation: A Study of Change over Twenty Years." Publisher: Taylor & Francis, Ltd. *The Journal of Higher Education* 71 (4): 454–475. ISSN: 0022-1546, accessed November 8, 2024. https://doi.org/10.2307/2649148. https://www.jstor.org/stable/2649148.

Minnotte, Krista Lynn. 2021. "Academic parenthood: Navigating structure and culture in an elite occupation." *Sociology Compass* 15, no. 7 (July): e12903. ISSN: 1751-9020, 1751-9020, accessed December 18, 2024. https://doi.org/10.1111/soc4.12903. https://compass.onlinelibrary.wiley.com/doi/10.1111/soc4.12903.

Nickel, Maximilian, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. "A Review of Relational Machine Learning for Knowledge Graphs." *Proceedings of the IEEE* 104, no. 1 (January): 11–33. ISSN: 0018-9219, 1558-2256, accessed December 12, 2024. https://doi.org/10.1109/JPROC.2015.2483592. arXiv: 1503.00759[stat]. http://arxiv.org/abs/1503.00759.

OpenAI. 2024a. "Advanced Usage." OpenAI Platform. Accessed November 23, 2024. https://platform.openai.com.

———. 2024b. "Models." OpenAI Platform. Accessed November 23, 2024. https://platform.openai.com.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. "Improving Language Understanding by Generative Pre-Training." *OpenAI,* accessed November 8, 2024.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. "Language Models are Unsupervised Multitask Learners." *OpenAI,* accessed November 8, 2024. https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,* arXiv:1910.01108, March 1, 2020. Accessed December 16, 2024. https://doi.org/10.48550/arXiv.1910.01108. arXiv: 1910.01108[cs]. http://arxiv.org/abs/1910.01108.

Sherman, Michael, Yuan Cao, Erick Armbrust, Anant Nawalgaria, and Antonio Gulli. 2024. "Prompt Engineering." *Google Whitepaper.*

Traag, V. A., L. Waltman, and N. J. Van Eck. 2019. "From Louvain to Leiden: guaranteeing well-connected communities." *Scientific Reports* 9, no. 1 (March 26, 2019): 5233. ISSN: 2045-2322, accessed December 12, 2024. https://doi.org/10.1038/s41598-019-41695-z. https://www.nature.com/articles/s41598-019-41695-z.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. *Attention Is All You Need,* arXiv:1706.03762, December 5, 2017. Accessed April 29, 2023. arXiv: 1706.03762[cs]. http://arxiv.org/abs/1706.03762.

Winslow, Sarah. 2010. "Gender Inequality and Time Allocations Among Academic Faculty." *Gender & Society* 24, no. 6 (December): 769–793. ISSN: 0891-2432, 1552-3977, accessed December 18, 2024. https://doi.org/10.1177/0891243210386728. https://journals.sagepub.com/doi/10.1177/0891243210386728.

A Default LLM Prompts

Below are the key components of each prompt for each respective module. These are used as the human prompts, with system prompts setting up basic personae (e.g. a college professor preparing for the user-provided course name).

Code Chunk 8: BeamerBot Prompt

```
"""
## Create a LaTeX Beamer presentation following the below guidelines:

### Source Documents and Examples

1. **Lesson Objectives**:
   - We are on lesson {lesson_no}.
   - Ensure each slide works toward the following lesson objectives:
   {objectives}

2. **Lesson Readings**:
   - Use these readings to guide your slide content:
   {information}

---

### General Format to Follow:

1. **Title Slide**:
   - Copy the prior lesson's title slide, **include author and institution
       from the last presentation**.

2. **Where We Are in the Course**
   - Last time: <Title of last lesson>
      - The readings from last lesson (Lesson {prior_lesson}).
      - **Include every assigned reading from this lesson.**
   - Today: <Title of the current lesson>
      - The readings for the current lesson (Lesson {lesson_no}).
      - **Include every assigned reading from this lesson**

3. **Lesson Objectives**:
    - The action in each lesson objective should be bolded (e.g. '\\textbf(
       Understand) the role of government.')

4. **Discussion Question**:
   - Add a thought-provoking question based on lesson material to initiate
       conversation.

5. **Lecture Slides**:
   - Cover key points from the lesson objectives and readings.
   - Ensure logical flow and alignment with the objectives.

6. **In-Class Exercise**:
   - Add an interactive exercise to engage and re-energize students.
   - This exercise should occur about halfway through the lecture slides, to
       get students re-engaged.

7. **Key Takeaways**:
   - Conclude with three primary takeaways from the lesson. These should
       emphasize the most critical points.
   - Bold or italicize the key terms for emphasis.

```

```
8. **Next Time**:
   - Provide the title of the next lesson (Lesson {lesson_no + 1}).
   - Include the assigned readings for the next lesson (Lesson {lesson_no + 1}).

---

### Specific guidance for this lesson:

{specific_guidance}

---

### Example of Expected Output:
    % This is an example format only. Use the provided last lesson as your primary source.
    % Replace the example \\author{{}} and \\institute{{}} below with the corresponding values from last lesson's presentation
    \\title{{Lesson 5: Interest Groups}}
    \\author{{}}
    \\institute[]{{}}
    \\date{{\\today}}
    \\begin{{document}}
    \\section{{Introduction}}
    \\begin{{frame}}
    \\titlepage
    \\end{{frame}}
    ...
    \\end{{document}}


{additional_guidance}

---

### Example of previous presentation:
- Use the presentation from last lesson as an example for formatting and structure:
{last_presentation}

---

### IMPORTANT:
- Use valid LaTeX syntax.
- The output should contain **only** LaTeX code, with no extra explanations.
- Start at the point in the preamble where we call \\title.
- Failure to follow the format and style of the last lesson's presentation may result in the output being rejected.
- Use the **same author and institute** as provided in the last lessons presentation. Do not invent new names or institutions. Copy these values exactly from the prior lesson.
- If unable to identify the author and institute from the last lesson, just leave them blank.
- Failure to follow these instructions will result in the output being rejected.
"""
```

Code Chunk 9: ConceptWeb Prompt for Summarization

```
"""
Create a structured summary of the following text, maintaining its academic depth and theoretical nuance.

### Summary Requirements:
- Length: 250-350 words
- Structure your summary with these components:
    1. **Central Thesis**: Primary theoretical argument or scholarly contribution
    2. **Theoretical Framework**: Key theoretical constructs and their relationships
    3. **Supporting Arguments**: Major lines of reasoning and evidence
    4. **Key Findings/Examples**: Empirical evidence or illustrative cases

### Important Considerations:
- Use clear, direct language
- Maintain explicit connections between concepts (e.g., "X leads to Y", "A shapes B")
- Include relevant scholarly context

**Text to Summarize:**
---
{text}
---

{additional_guidance}
"""
```

Code Chunk 10: ConceptWeb Prompt for Relationship Extraction

```
"""
## You will analyze the text for this lesson to extract *key concepts* and their *relationships* in light of the lesson objectives.

### Lesson Details:
- **Objectives**:
    {objectives}

### Task Instructions:
From the text provided, identify:
1. **Core Theoretical Concepts** (e.g., "State of Nature", "Social Contract")
2. **Causal Mechanisms** (how concepts influence each other)
3. **Structural Relationships** (how concepts fit together)

### Concept Extraction Guidelines:
- Focus on concepts and relationships present in the source text
- Relationships must be directly derived from the source text:
    - Avoid introducing external historical references unless explicitly mentioned in the text.
- Focus on both:
    - Theoretical concepts or important themes (e.g., "Social Contract", "Sovereignty", "Separation of Powers")
    - Causal mechanisms (e.g., "Human Nature", "War", "Competition")
- Include intermediate concepts that connect major ideas
- Ensure concepts form complete theoretical chains
- Each concept should be a significant theoretical idea or mechanism. Examples:
    GOOD: "Separation of Powers"        (fundamental principle)
    BAD:  "House of Representatives"    (too specific: instance of representation)
    GOOD: "Social Contract"             (key theoretical concept)
    BAD:  "Hobbes Leviathan Chapter 2"  (source rather than concept)
    GOOD: "Natural Rights"              (broad theoretical principle)
    BAD:  "Right to Bear Arms"          (specific instance of rights)
- Limit concepts and relationship terms to **no more than three words each** for clarity.

### Relationship Mapping Guidelines:
- Use precise relationship verbs that show:
    - Causation ("leads to", "creates", "produces")
    - Definition ("comprises", "consists of", "characterized by")
    - Support or theoretical connections ("enables", "maintains", "preserves")
- Ensure relationships form complete theoretical pathways

Structure relationships in the format:
‘‘‘json
"relationships": [
    ["Concept 1", "relationship_type", "Concept 2"],
    ["Concept 1", "relationship_type", "Concept 3"],
    ...
]
‘‘‘

{additional_guidance}

### IMPORTANT: Your final response must strictly follow this JSON format:
‘‘‘json
{{
    "concepts": [
    "Concept 1",
```

```
    "Concept 2",
    "Concept 3",
    "Concept 4",
    ...
    ],
    "relationships": [
    ["Concept 1", "relationship_to_Concept_2", "Concept 2"],
    ["Concept 1", "relationship_to_Concept_3", "Concept 3"],
    ["Concept 1", "relationship_to_Concept_4", "Concept 4"],
    ["Concept 2", "relationship_to_Concept_3", "Concept 3"],
    ["Concept 2", "relationship_to_Concept_4", "Concept 4"],
    ...
    ]
}}
```

```
### REMINDER:
- Ensure your final response strictly adheres to the JSON format provided. Include only the json in your response.
- If the JSON is invalid or extra text is included, your response will be rejected.
"""
```

Code Chunk 11: QuizMaker Prompt

```
"""
## Generate a quiz according to the following guidelines

### Context and Objectives:
- The quiz should cover major ideas from the lesson objectives listed here:
{objectives}

- Here are the texts that the quiz will cover: {information}

---

### Task Instructions:
Generate **9 quiz questions** using the following breakdown:
1. **3 Multiple-choice Questions**
   - Provide 4 answer choices (A, B, C, D) for each question.
   - Ensure the correct answer is explicitly stated in the `"correct_answer"` field (e.g., "A)", "B)", "C)", or "D)").
   - Balance correct answers across all options (A, B, C, D should each be used once across the entire quiz).

2. **3 True/False Questions**
   - Format each question with "True" (A) and "False" (B) as answer options.
   - Ensure `"correct_answer"` is clearly identified as either "A" or "B".
   - "C" and "D" should return empty strings.

3. **3 Fill-in-the-blank Questions**
   - Provide the question with a blank to be filled in.
   - Include **plausible answer choices** (A, B, C, D), with the correct answer explicitly stated in the `"correct_answer"` field.


---

### Difficulty Level:
Generate questions at a difficulty level of {difficulty_level} on a scale of 1 to 10:
- **1-3 (Easy)**: Simple recall of definitions or basic facts.
- **4-7 (Medium)**: Understanding relationships between concepts.
- **8-10 (Hard)**: Application and analysis, requiring synthesis of multiple ideas.

---

### Output Format:
Here is an example output format. Your response **must** strictly follow this format:

    {{
      "multiple_choice": [
        {{
          "question": "Question text here",
          "A)": "Choice 1",
          "B)": "Choice 2",
          "C)": "Choice 3",
          "D)": "Choice 4",
          "correct_answer": "A, B, C, or D corresponding to the correct choice"
        }},
        ...
      ],
```

```
    "true_false": [
      {{
        "question": "True/False question text here",
        "A)": "True",
        "B)": "False",
        "C)": "",
        "D)": "",
        "correct_answer": "A or B"
      }},
      ...
    ],

    "fill_in_the_blank": [
      {{
        "question": "Question text here",
        "A)": "Choice 1",
        "B)": "Choice 2",
        "C)": "Choice 3",
        "D)": "Choice 4",
        "correct_answer": "Correct answer that completes the missing words"
      }},
      ...
    ]
  }}

---

### Important Notes:
- **JSON Only**: Include only the JSON structure in your response.
- **Correct Answer Balance**: Ensure that correct answers (A, B, C, D) are distributed evenly across all multiple-choice questions.
- **Plausible Choices**: For all questions, especially fill-in-the-blank, ensure distractor choices are realistic and relevant to the topic.
- **Avoid Duplication**: Do not include any overlap with these existing questions:
  {prior_quiz_questions}


{additional_guidance}

---

### Reminder:
- Responses **must** adhere strictly to the JSON format provided above.
- Extra text, incorrect formatting, or invalid JSON will result in rejection.
"""
```

B Example Ad Hoc LLM Query and Reply

The below shows the results of a simple query such as we might expect a standard user to prompt an LLM when creating lesson materials. While the slide *content* was of a generally acceptable quality for an instructor to start building from (e.g. extracts were quoted correctly), the *form* of the response was lacking. The generated slides would not compile correctly in a LaTeX environment due to missing requirements in the LaTeX preamble (this example used Overleaf as a compiler. The attempted compile generated 130 errors). When this was pointed out to the model, it attempted multiple times to fix its code errors, but was unsuccessful.

Human Prompt:

```
"I need to create a LaTeX beamer slideshow for the class on American
    politics I'm currently teaching. Here are the readings. We'll be
    discussing Locke and Montesqueiu. Also attached are the lesson
    objectives. We're on Lesson 3
```

Model Response [GPT-4o]:

```
I will create a LaTeX Beamer presentation for Lesson 3 on "Philosophical
    Influences of the Founders," focusing on Locke and Montesquieu. The
    presentation will incorporate the objectives and key ideas from the
    provided readings.

\documentclass{beamer}

% Presentation Info
\title[Lesson 3: Philosophical Influences]{Philosophical Influences of the
    Founders}
\subtitle{Lesson 3: American Politics}
\author{Your Name}
\institute{Your Institution}
\date{\today}

% Begin Document
\begin{document}

% Title Slide
\begin{frame}
    \titlepage
\end{frame}

% Slide: Lesson Objectives
\begin{frame}{Lesson Objectives}
    \begin{itemize}
        \item Describe Locke's views on human nature, the purpose of
            governments, and how governments form.
        \item Explain Montesquieu's definition of liberty and his views on how
             liberty is best preserved.
    \end{itemize}
\end{frame}

% Slide: Introduction to Locke
\begin{frame}{John Locke: Second Treatise of Government}
```

```
    \begin{itemize}
        \item \textbf{Human Nature}: Naturally free, equal, and independent.
        \item \textbf{Purpose of Government}: To preserve life, liberty, and property.
        \item \textbf{Formation of Governments}: Based on the consent of the governed.
        \item \textbf{Key Idea}: Majority rule as the basis for collective decisions.
    \end{itemize}
    \vspace{1em}
    \textit{Quote:} \textit{\"The only way whereby anyone divests himself of his natural liberty, and puts on the bonds of civil society, is by agreeing with other men to join and unite into a community.\"} (Sect. 95)
\end{frame}

% Slide: Locke's Ideas on Governance
\begin{frame}{Key Features of Locke's Political Philosophy}
    \begin{itemize}
        \item \textbf{Natural Rights}: Life, liberty, property.
        \item \textbf{Tacit Consent}: Living under a government implies agreement to its laws.
        \item \textbf{Separation of Powers}: Importance of distinct legislative and executive powers.
        \item \textbf{Purpose of Laws}: To establish fairness and protect individuals.
    \end{itemize}
\end{frame}

% Slide: Introduction to Montesquieu
\begin{frame}{Montesquieu: The Spirit of the Laws}
    \begin{itemize}
        \item \textbf{Definition of Liberty}: Not unlimited freedom but acting within the bounds of laws.
        \item \textbf{Preservation of Liberty}: Requires separation of powers.
        \item \textbf{Types of Government}: Republican, Monarchical, Despotic.
        \item \textbf{Role of Education}: Instill love for laws and the public good in republics.
    \end{itemize}
    \vspace{1em}
    \textit{Quote:} \textit{\"Liberty is a right of doing whatever the laws permit.\"} (Book XI, Chapter III)
\end{frame}

% Slide: Montesquieu's Separation of Powers
\begin{frame}{Separation of Powers}
    \begin{itemize}
        \item \textbf{Three Powers of Government}:
        \begin{itemize}
            \item Legislative
            \item Executive
            \item Judiciary
        \end{itemize}
        \item \textbf{Key Principle}: Power should check power to prevent abuse.
        \item \textbf{Application}: Liberty exists when these powers are separated and balanced.
    \end{itemize}
\end{frame}

% Slide: Comparing Locke and Montesquieu
\begin{frame}{Comparison: Locke vs. Montesquieu}
```

```
    \begin{tabular}{|c|c|c|}
        \hline
        & \textbf{Locke} & \textbf{Montesquieu} \\
        \hline
        \textbf{Human Nature} & Free and equal & Shaped by education \\
        \hline
        \textbf{Purpose of Government} & Protect property & Preserve liberty \\
        \hline
        \textbf{Key Idea} & Consent of governed & Separation of powers \\
        \hline
    \end{tabular}
\end{frame}

% Slide: Discussion Questions
\begin{frame}{Discussion Questions}
    \begin{itemize}
        \item How do Locke's ideas on property influence the American political system?
        \item Can Montesquieu's separation of powers prevent all forms of tyranny?
        \item How do these philosophies shape contemporary American government?
    \end{itemize}
\end{frame}

% End Slide
\begin{frame}
    \centering
    \Huge{Thank You!}
    \vspace{1em}
    \normalsize Questions or Comments?
\end{frame}

\end{document}

I've created a LaTeX Beamer presentation for Lesson 3, focusing on Locke and Montesquieu. Let me know if you want any adjustments or additional slides!
```