LLM Agents: Large Language Models as Actors in Text-Based Environments

Jonathan Koch jkoch21@usf.edu
Fiorella Andrea Ratti Tamayo frattitamayo@usf.edu
Juan Gomez jjgomez@usf.edu
John Licato licato@usf.edu

June 23rd, 2023

Abstract

Recent advancements in Large Language Models (LLMs) have enhanced capabilities in generative text, problem-solving, and situational awareness. These developments, highlighted in studies by Wang et al. Wang et al. [2023] and Xi et al. Xi et al. [2023], reveal emergent properties in system-prompted responses. We build on the concept of LLMs as autonomous agents Fan and Anandkumar [2023], exploring their cognitive abilities Dhingra et al. [2023]. Our hypothesis, drawing from Kolmogorov Complexity Kolmogorov [1968] and recent insights into LLMs Wei et al. [2022], suggests that non-finetuned LLMs can adaptively modify environments for specific problems. This paper presents the "Teach-a-Bull AI Tutor" environment. It uses LLMs as Actors in Text-Based Environments (LLMaAiT-BE) to study their generative capabilities in virtualized, automated education. Our research aims to understand LLMs' role in education, as proposed by Park et al. Park et al. [2023]. We focus on developing a formal method for designing cognitive processes with LLMs, targeting educational applications.

---

1 Introduction

The rapid evolution of Large Language Models (LLMs) like GPT-4 marks a significant milestone in the realm of artificial intelligence, particularly in their application as autonomous agents in complex environments. The development of the Transformer architecture has been pivotal in this advancement, enabling models to exhibit emergent properties such as creative writing, efficient information recollection, and enhanced chatbot functionalities, as seen in projects as seen in projects such as Bard by Google and ChatGPT by OpenAI Wei et al. [2022] Wang et al. [2023]. Recent explorations into the capabilities of LLMs have shown promising results, extending beyond conventional text generation to more interactive and dynamic applications. For instance, the Voyager project demonstrated the ability of an LLM to learn and adapt within the Minecraft environment, exhibiting novel problem-solving skills in a virtual world Fan and Anandkumar [2023]. Similarly, the concept of Generative Agents has been introduced, showcasing how LLMs can simulate human-like behavior and interactions within a given context, providing insights into their potential as intelligent, responsive entities in digital environments Park et al. [2023].

This paper aims to contribute to this growing field of research by presenting Large Language Models as Actors in Text-Based Environments (LLMaAiT-BE). Our focus is on harnessing the reasoning and generative capabilities of non-finetuned LLMs to interact within a specifically designed text-based environment. The "Teach-a-Bull AI Tutor" environment is a prime example of such an application, where the LLMs' ability to construct a learning plan, generate educational content and engage in interactive learning scenarios will be explored and evaluated.
By integrating concepts from cognitive psychology and AI, such as the works of Dhingra et al. (2023) and Kolmogorov (1968), we seek to understand and utilize the cognitive processes underlying these models, especially in the context of pattern matching and decision-making. The research outlined here does not assert a definitive structure for educational methods but rather uses LLMs as a tool to explore new possibilities in virtualized and automated education Dhingra et al. [2023] Xi et al. [2023].
Through this exploration, LLMaAiT-BE aims to provide a deeper understanding of the capabilities of LLMs in interactive and adaptive environments, potentially revolutionizing the way we perceive and utilize AI in educational and other text-based settings.

2 Motivations

2.1 Foundations of Cognitive Abilities in LLMs

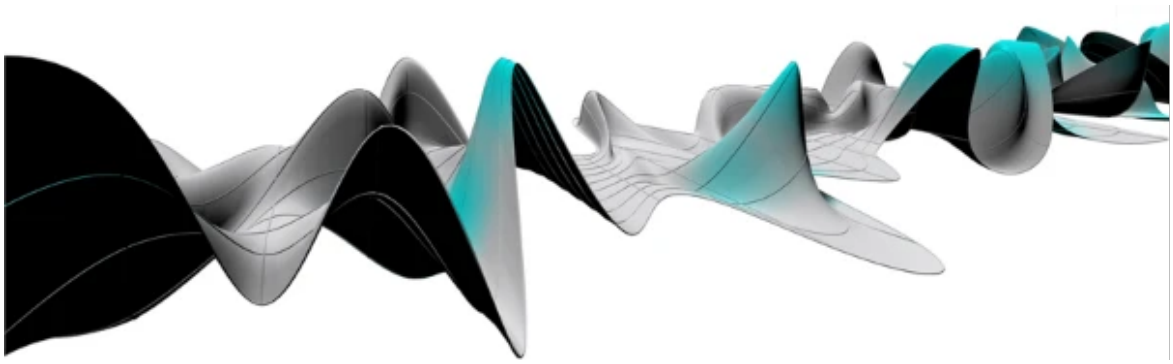

Figure 1: A first image of natural language in 3D. From: Natural language modelled and printed in 3D: a multi-disciplinary approach, Fig. 5. Formulated as a digitally woven fabric.
Pillen and Matthews [2022]

Patterns in Data and Algorithms Algorithms are fundamental to the functioning of Large Language Models (LLMs) as they enable the recognition and utilization of patterns within data. To compute an output from a given input, programmers must consider several factors. The entropy of the input data is a measure of its randomness or unpredictability, which affects how the algorithm processes the data. Additionally, understanding the probability distribution from which the data originates helps in predicting the likelihood of various data occurrences. Lastly, recognizing structural patterns within the data is crucial, especially when you are actively conditioning on input. You can think of a simple example where you are playing a color memorization game. After playing several games, you notice that during the games you've played, green tends to occur right after blue exclusively. Contrary to your initial biases, you should be able to recognize this as a pattern and actively adapt. These patterns exist in language, whether in language syntax, semantic relationships, or contextual cues. Many speculate that the models are finding these patterns actively from the input data as a guide for generating accurate and coherent responses.

To depict these patterns visually, *Figure 1* 1 demonstrates the relationship between syllables per line (y axis), Evident Weight (z axis), and Time (x axis). Although a simple pattern, *Pillen*Pillen and Matthews [2022] details how these patterns change dramatically depending on the dialect and language of the input data. Such insights underscore the importance of identifying and understanding these distributions, particularly in unsupervised learning contexts, where they hold the key to unlocking meaningful representations from seemingly abstract data.

Kolmogorov Complexity and Compression The conceptual framework of Kolmogorov Complexity, centering on the aspect of data compression, offers profound insights into the cognitive functions of Large Language Models (LLMs). At its core, this principle involves a hypothetical perfect compressor, K, which for any given data set X, can output the most concise program possible. This ideal compression scenario, as formulated by KolmogorovKolmogorov [1968], suggests that K effectively exploits all recognizable patterns in X. The equation representing this concept is:

$$K(X) \leq |C(X)| + K(C) + O(1) \tag{1}$$

Here, C signifies any alternative compressor, indicating that the length of the compressed output of X by C is a combination of the length of C's output, the Kolmogorov Complexity of C, and a constant. This relationship forms a crucial understanding of how compression interlinks with algorithmic complexity and pattern recognition in dataKolmogorov [1968]. Furthermore, when considering the concatenation of two datasets X and Y, the compression process can be represented as:

$$K(X,Y) = K(X) + K(Y|X) + O(log(K(X|Y))) \tag{2}$$

This formula suggests that the perfect compression of $CONCAT(X,Y)$ involves leveraging patterns across both datasets. Ilya Sutskever's observation, "First generate X, then use X to generate Y, and this can't be too different than generating the two datasets jointly"Sutskever [2023], underlines the significance of joint pattern extraction in the compression process. This concept is pivotal in understanding how LLMs can actively learn how to solve problems, and even mimic cognitive processes for efficient data processing and pattern utilization. A good compressor would utilize all of the patterns from the concatenation data to better compress the two jointly as opposed to independently. We mention this to bring attention to the emergence of many desirable cognitive properties, those of which many humans would argue demonstrate intelligence. These properties, which emerge primarily through compression, provide us useful insight in how to prompt LLMs in a manner that maximizes their ability to complete our desired tasks.

Language Models as Compressors and Predictors In the realm of Large Language Models (LLMs), their operation can be conceptualized as a sophisticated form of data compression, fundamentally aligned with the principles of Maximum Likelihood Estimation (MLE). MLE serves as a cornerstone in optimizing these models, essentially focusing on the probability maximization of observing the given data. Mathematically, for a set of observed data $\mathbf{X}$, the MLE aims to find the model parameters θ that maximize the likelihood function $L(\theta|\mathbf{X})$, *i.e.* find the set of parameters that maximize the likely-hood of the training data. This can be expressed as:

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}}\ L(\theta|\mathbf{X}) \tag{3}$$

Here, $\hat{\theta}_{MLE}$ represents the parameter values that maximize the likelihood function, essentially compressing the information in $\mathbf{X}$ into a model with optimal predictive capabilities. This perspective positions LLMs as not just generators of text, but as entities that encode and compress vast quantities of linguistic data, enabling them to predict or generate coherent, contextually appropriate responses. This dual role as compressors and predictors highlights the intricate balance LLMs maintain between data encoding and generative output, embodying the essence of advanced AI-driven linguistic processing.

A notable aspect is the one-to-one correlation between compressors and predictors, positing that enhanced data compression equates to improved prediction accuracy. Under the lens of Kolmogorov Complexity, LLMs like GPT-3 and LLaMa, optimized via Stochastic Gradient Descent (SGD), undertake a task akin to $K(X, Y)$, compressing diverse internet documents. This optimization suggests that the most effective LLM for a given document D is also its best compressor, learned through SGD. This process entails learning every significant pattern in D's distribution, enabling the model to sample from a probability distribution P for a word W given a partially completed document D'. Therefore, if D' embodies intelligent or meaningful content, it allows sampling from a distribution that could be perceived as intelligently representative of language.

"I think, therefore I am."
René Descartes

The Role of Language in Intelligence *"Why Language?"*, Daniel Dennett's insights on language and intelligence suggest that language is a critical component of our cognitive system, acting as a complex multi-dimensional graph that forms the basis for our thoughts and ideas. He states that our uniqueness in intelligence as humans stems from its capability to *"[permit] our hypotheses to die in our stead"* Dennett [2015]. This complexity is mirrored in LLMs, where language processing involves identifying and leveraging patterns within data. We can demonstrate that one may be able to effectively guide a model to a better answer, in fact many recent papers such as Chain of Thought Wei et al. [2023] or Tree of ThoughtYao et al. [2023] demonstrate a models ability to reason can be used to improve its ability to predict. Research indicates that the size of the model, such as in GPT-4 and LLaMa, plays a significant role in their ability to exhibit emergent properties and perform cognitive tasks effectively Wei et al. [2022].

The advancement of LLMs, particularly in handling complex language structures and adapting to new information, parallels the human cognitive system's reliance on language for thought and reasoning. This parallel underscores the potential of LLMs in mimicking human-like intelligence and decision-making processes.

Cognitive Processes in LLMs Many would suggest that although we have some very sophisticated computer algorithms that we can develop with formal logic, many of the most interesting problems, such as object classification, autonomous driving systems, Text-to-Speech to name a few, have no clear connection from input data being processed by an computer algorithm to optimal output data. However, there exist patterns in higher-level dimensions which an optimal compression mechanisms can exploit to perform similar tricks to which the language models exhibit. We propose that LLMs exhibit pattern recognition and exploitation, what we will refer to as cognitive processing skills, or, they have the ability to perform a cognitive process. We are implying that a cognitive process is an algorithmic-like pattern matching and decision-making problem which is not necessarily easy to develop a logical (typed) algorithm for, such as predicting the sentiment of

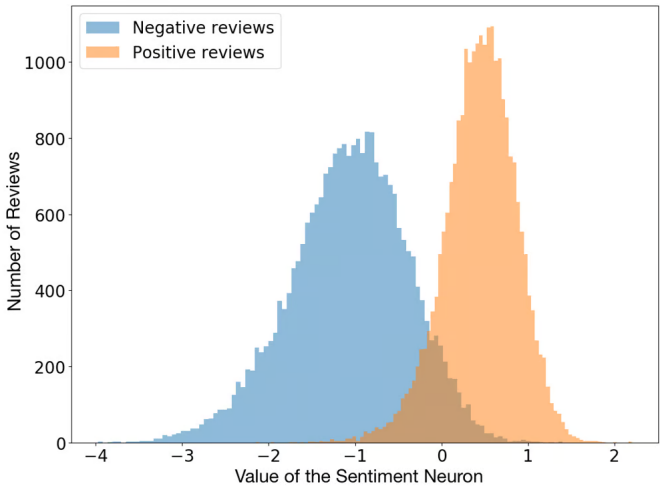

Figure 2: Unsupervised learning demonstrates the activation's of a Neuron's activation correlates to the sentiment of the movie review.

Radford et al. [2017]

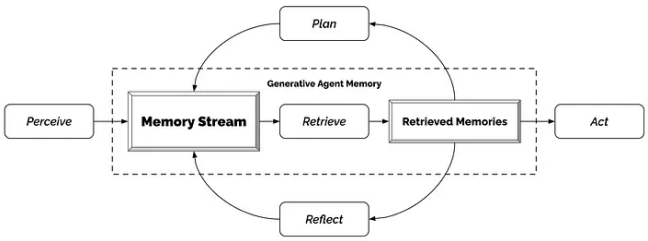

Figure 3: Agents perceive their environment, and all perceptions are saved in a comprehensive record of the agent's experiences called the memory stream. Based on their perceptions, the architecture retrieves relevant memories and uses those retrieved actions to determine an action. These retrieved memories are also used to form longer-term plans and create higher-level reflections, both of which are entered into the memory stream for future use.

Park et al. [2023]

am Amazon Review. Research out of OpenAI demonstrated that unsupervised learning on movie reviews for next char/byte prediction leads to the emergent "Sentiment Neuron" Radford et al. [2017], where for most of the dataset the activation of the neuron corresponds to the sentiment of the document. For many this may seem surprising, however one question I pose is, "How is it any different of a representation is a neural network's learned representation for sentiment than for humans representation?". Our research looks to dive into the emergent properties expressed by Large Language Models (LLMs) and their abilities to perform these cognitive processes involving pattern matching, information generation, and decision-making.
Large Language Models (LLMs) like GPT display a range of cognitive processes, akin to algorithmic pattern matching and decision-making, which can be challenging to replicate with conventional logical algorithms without a world model.

- "Generative Agents: Interactive Simulacra of Human Behavior," LLMs demonstrate capabilities such as coordination and event planning, novel agent architecture 3, memory and retrieval, reflection, planning and reaction, and dialogue generation Park et al. [2023]. These abilities illustrate how LLMs can handle complex tasks, making them effective in diverse scenarios.

- "Voyager: Minecraft GPT-4 Agent" showcases GPT-4's cognitive skills in planning, execution, tool creation, action generalization, and error correction. It demonstrates continuous learning and adaptation, novel problem-solving, autonomous decision-making, and zero-shot generalization on unseen tasks, underscoring its adaptability and learning capabilities Fan and Anandkumar [2023].

- "Dhingra et al. (2023)" further explores GPT-4's cognitive psychology, highlighting common sense reasoning, language processing, creativity, and analogical reasoning. This research indicates that GPT-4 can integrate various cognitive skills to process and generate human-like responses Dhingra et al. [2023].

A Simplified Model of Cognitive Processes We propose simplifying these cognitive abilities into three key tasks; planning, translation, and discrimination. Planning involves generating information relevant to the environment, while translation translates plans into actionable structures. Discrimination involves decision-making based on the current state and potential future states of the environment. These make up what we refer to as **Cognitive Tasks**:

- **Planning**: also framed as information generation surrounding a problem, planning provides us with natural language insight around maneuvering a state in the environment. We use planning for; (a) filling out a data structure with contextual information based on the current state of the environment; (b) Memory and Planning modules used for future reference – this is akin to reflection and memory modules used in Generative Agents Park et al. [2023], (c) reasoning about which actions to take and how those actions will modify the current state of the environment, and (d) generating informational context around a stimulus. In the Teach-A-Bull environment that we have developed, we use planning during almost every phase of information processing. For example, we plan how to respond to a Student querying help with some subject. We can think of planning as encompassing many of the Agent Architecture such as planning and reflecting. Planning encompasses creative problem-solving and creative generation.

- **Translation**: used for translating data into a format in which we can execute on in our program, e.g. we translate a natural language action plan into an action JSON object which we extract from the LLM's outputs.

- **Discrimination**: based on the environment state (i.e. the current generation state, current generation action, ...) discriminate if; (a) Is the current action good? *(i.e. does this generation pass a certain criteria)*, (b) Is the current state optimal? (i.e. we can terminate the current process if the state is optimal or the state is a termination state), (c) What information is relevant for future states? *(i.e. memory/selective information passing)*.

Note that our model removes a specifically defined perception module, and instead we incorporate perception into each of the three previous tasks; as the environment state is always to some degree partially observable by the LLM agent. To do so, we include details related to what information it is presented with such that it can passively perceive the state during all cognitive tasks.
We propose a new simplified model under which we will detail how to formulate any text-based process under; these we refer to as **Cognitive Processes**. A cognitive process is that which can emulate cognitive abilities which humans may also demonstrate, such as teaching, developing and potentially even eventually researching. As we are dealing with only the text based modality, we are unable to reflect on the transfer of our formulation to other modalities, however it is evident that it should also be possible. *Dennet* proposes that

*"... our brains are in effect joined together into a single cognitive system that dwarfs all others. They are joined by one of the innovations that has invaded our brains and no others: language"*Dennett [2015].
At the center of this we can assume that the ability to use and manipulate language is an expressive bridge between the objective and subjective worlds and experiences. Based on this, we can develop a text-based environment built on top of a collection of inter-connected Cognitive Processes built on predicting and parsing language; e.g. for completion of a Slide, which we will detail further, we can process the state in a planning module, and pass the result into a translation module. When we refer to a cognitive process, we are referring to the piecing together of cognitive tasks such that the input from one cognitive task relies on the output of the previous.

3 Methodology

3.1 Documents and Generation Objects

In our research, we have focused on the generation of objects that are contextually relevant to specific environments. For instance, in the Teach-A-Bull environment, the generation targets teaching-related content, although this model is versatile enough to be applied to diverse domain, such as programming, software documentation, and academic research. This flexibility is achieved through a comprehensive model that utilizes cognitive processes for document generation.

Notebank The Notebank serves as a crucial planning and memory module, conditioning the environment according to the student's learning goals, special instructions, and perspective. It functions as a repository of JSON-formatted action schemes, facilitating efficient interaction between the tutor and the learning environment.

Concepts and Concept Graph Concepts are pivotal to understanding complex topics, acting as nodes in a knowledge network. Each concept is interlinked with others, creating a web of information essential for learning. For example, the mathematical concept of a "derivative" connects to "function", "input," and "output," building a comprehensive understanding of the term. These interlinked concepts form the backbone of our educational document generation process, similar to methodologies discussed in contemporary knowledge generation research. The Concept Data

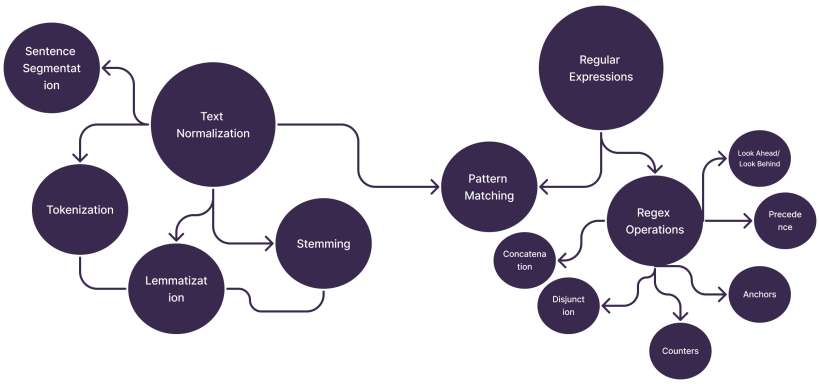

Figure 4: Visual Representation of a Concept Graph

Structure includes the concept's name, definition, related concepts, and an optional LaTeX code. Our prompting technique combines planning, discrimination, and translation, ensuring a coherent and contextually relevant generation of concept objects.
The Concept Graph 4 represents the knowledge structure within the Learning Environment. Our LLM agent is adept at identifying relevant concepts and mapping the interconnections within this graph. The graph is centered around a 'Main Concept,' influenced by the student's interactions with the Tutor. This Main Concept acts as the focal point of the learning journey, guiding the discovery of related topics through structured prompting. The use of a Concept Graph facilitates a unique generative pattern, initiating from the Main Concept and expanding through a 'show your work' approach in prompting. This methodology not only enhances understanding but also promotes a deeper exploration of the subject matter.

SlidePlans and Slides The Slides Data Structure are the most important and greatest challenge to effective learning, essentially, we must device a cohesive sequence of presentation slides which completely covers the set of knowledge in which our student would be learning from. As a Data Structure, we came up with a method to enhance the flow of a slide deque such that it is more broad and cohesive which plays further into the Markov Assumption Markov [1954]. First, it is important to understand that our slide builds on top of the concepts, similar to the questions, which requires a slide to map to N number of concepts in our Concept Graph. By mapping a slide to a Concept C, the explored state of C will increment in the observations of the environment. We condition the Tutor to (a) map to each discovered concept in the Concept Graph at least once, (b) iterate on concepts with the slides i.e. we should explore a concept and then use it in a more applicable fashion, and (c) keep track of the relevant information which will need to be mapped to by the slide. Our slide contains purposes, which we define to be **Introductory**, **Relative**, **Exploratory**, **Explanative**, and **Examplative**. We encourage at each state s_t, we reflect on the current flow of information through the slide deque; introduce concepts which haven't been discussed, relate important concepts which we have discovered, explore new ideas surrounding a concept we have already discovered, explain further some idea in which we may have already learned, and provide an example or problem for which the student can be guided through. A purpose statement is also included in a Slide Plan, where its purpose is to later help the tutor gen-

erate the actual content being mapped to on the slide. In order to terminate the Markov Process, the Tutor will discriminate the current state of the environment and evaluate whether or not it should continue developing new slides or if is in an adequate state to start generating the slides. When generating a slide, we develop a $1-1$ process where each slide plan will become a slide. The slide contains three additional things which will need to also be generated; the slide description, content, and presentation. The description is a fully detailed knowledge base of information which the tutor will use to pull information from during slide generation as well as education with the student as a reference. The content is the data which the slide will contain, the presentation is the spoken blurb which the Tutor will use directly to teach the student, essentially it is the starting point of a slide's potential conversational window.

Example What a Slide Looks Like:


```
{
    "slides": [
        ...,
        {
            "title": "Discovering Lemmatization: Enhancing Text
               Analysis in NLP",
            "presentation": "Okay, let's delve into the world of
               lemmatization. Imagine we've got a pile of words in
               front of us, all different shapes and forms but with a
               shared identity. Think of 'run', 'runs', 'ran', and '
               running'. Essentially, they're all the same action,
               right? Well, in NLP, we use lemmatization to strip
               these words down to their core, the lemma, which in
               this case is 'run'. Now, this isn't just about cutting
               corners or simplifying text; it's a meticulous process
               that respects the word's grammatical stance. Unlike
               stemming, which might just chop off word endings and
               call it a day, lemmatization considers how the word
               functions in a sentence, its actual meaning, and its
               inherent grammar. This is of tremendous help in NLP
               tasks, especially when the nuance of meaning is a game-
               changer, like in distinguishing verbs from nouns or
               adjectives in their comparative forms. By standardizing
                text this way, we're giving AI algorithms a clearer,
               more consistent diet of data to work with, enhancing
               machine learning model performance across various
               applications, including text indexing, topic modeling,
               and poetic semantic analysis. Yes, even poetry can be
               grist for the NLP mill. The process starts with
               tokenization, breaking down sentences into individual
               pieces, kind of like dissecting sentences into lego
               blocks. Then comes the cool part - morphological
               analysis. Here we're detective Holmes, examining how
               each word is built, its grammatical relationships, and
               then finally, we piece together a lemmatized sentence.
               Take the sentence, 'The striped bats are hanging on
               their feet for best.' Post lemmatization, we get a
```

```
            transformed lineup like 'The stripe bat be hang on
            their foot for good.' Moving forward, we'll see how
            this plays out in real-world scenarios, like chatbots,
            where understanding human language is critical.
            Lemmatization ensures these applications don't just
            hear, but they comprehend. It's like we're teaching
            them the difference between just nodding along and
            actually getting the point of the conversation. Now,
            any questions before we jump into some practical
            exercises?",
        "content": "Lemmatization is the process of reducing words
            to their base or dictionary form (lemma)\nExamples: '
            run', 'runs', 'ran', 'running' -> Lemma: 'run'\
            nLemmatization vs. Stemming:\n- Stemming: Cuts off the
            ends of words (often crudely)\n- Lemmatization:
            Considers word's grammatical role, meaning, and
            morphologies\nImportance in NLP:\n- Enhances text
            consistency and standardization for AI algorithms\n-
            Improves performance of machine learning models\n-
            Crucial for text indexing, topic modeling, semantic
            analysis\nWorks by:\n- 1. Tokenizing text\n- 2.
            Analyzing morphological structure\n- 3. Producing
            lemmatized output\nExample Process: 'The striped bats
            are hanging on their feet for best.'\nStep 1:
            Tokenization -> ['The', 'striped', 'bats', 'are', '
            hanging', 'on', 'their', 'feet', 'for', 'best']\nStep
            2: Morphological Analysis -> Identifying parts of
            speech and relationships\nStep 3: Lemmatization Output
            -> ['The', 'strip', 'bat', 'be', 'hang', 'on', 'their',
            'foot', 'for', 'good']\nNext steps: Integration of
            Lemmatization in practical NLP applications (e.g.,
            chatbots)",
        "latex_codes": "",
        "purpose": 0,
        "purpose_statement": "This slide aims to introduce
            Lemmatization as an essential NLP text preprocessing
            technique, building on the student's understanding of
            Text Normalization and Tokenization, to further enhance
            their text analysis skills.",
        "concepts": [
            "Lemmatization",
            "Text Normalization",
            "Tokenization of Text"
        ]
    },
    ...
]
}
```

Questions In the development of the Teach-A-Bull environment, the Questions Data Structure plays a pivotal role. It is designed with a focus on adaptability, catering to various subjects and question types, each with unique requirements and specifications. As a student with a wide range of possible learning goals and outcomes, a generative system needs to be able to adapt adequately to the needs and desires of the student.
The Subject and Type are critical components, guiding the structure and content of questions. Subjects like **math**, **coding**, **literature**, and **conceptual** dictate the thematic focus and content of questions, while types like **free response** questions, **multiple choice**, **calculation/math** questions, and **coding problems** define their interactive format and grading style. These ensure that each question is contextually aligned with its intended educational objective, and that we can verify the integrity of user response to provide them adequate feedback through our education model. The following LLM-generated questions demonstrate the differences in how the Tutor can create questions, and are also example generations that the Tutor created during our testing:

Example 1. Code Entry Question for Prime Number Checking:

```
{
  "subject": 1,  // Code
  "type": 3,     // Code Entry
  "data": "Write a function to check if a number is prime.",
  "boilerplate": "def is_prime(x):\n    pass # Implement Function
     Logic",
  "test_cases_script": "assert is_prime(5) == True; assert is_prime(4)
      == False;",
  "concepts": ["Function", "Conditional Statements"]
}
```

Example 2. Calculation Entry Question for calculating Derivative:

```
{
  "subject": 0,  // Math
  "type": 2,     // Calculation Entry
  "data": "What is the derivative of the following expression when x
     =6?",
  "latex_code": "frac{d}{dx} (x^2) - 2",
  "calculation_script": "import sympy as sp; x = sp.symbols('x'); expr
      = x**2 - 2; deriv = sp.diff(expr, x); result = deriv.subs(x, 6);
      print(result);",
  "concepts": ["Derivatives", "Polynomials"]
}
```

Example 3. Conceptual Free Response Question Comparing and Contrasting Machine Learning and Deep Learning:

```
{
"subject": 3,  // Conceptual
"type": 0,     // Text Entry
"data": "Compare and Contrast 'Machine Learning' and 'Deep Learning'.
   [3-5 sentences]",
"rubric": "Rubric: [1 Points] Student has at least 3 sentences. [2
   points] All information provided is academically accurate. [1
```

```
    points] Student compares and contrasts Machine Learning and Deep
    Learning",
"concepts": ["Artificial Intelligence", "Machine Learning", "Deep
    Learning"]
}
```

Example 4. Literature Multiple Choice Question about a sample Reading Passage:

```
{
  "subject": 2,  // Literature
  "type": 1,     // Multiple Choice
  "data": "In the passage, how does the author convey the theme of
     perseverance against adversity?",
  "reading_passage": "Amidst the roaring tempest, Eleanor clutched the
      tattered photograph to her chest. The storm had taken everything
     , yet here she stood at the cliff's edge, eyes blazing with
     determination. 'This tempest can't erode my spirit,' she thought.
      'For in my heart, the flame of hope burns eternal.'",
  "entry_1": "Through the symbolism of the storm",
  "entry_2": "By illustrating Eleanor's emotional resilience",
  "entry_3": "By depicting the loss of material possessions",
  "entry_4": "All of the above",
  "correct_entry": "entry_2",
  "concepts": ["Reading Comprehension", "Literary Techniques", "Theme
     Analysis"]
}

{
  "subject": 2,  // Literature
  "type": 1,     // Multiple Choice
  "data": "In the passage, how does the author convey the theme of
     perseverance against adversity?",
  "reading_passage": "Amidst the roaring tempest, Eleanor clutched the
      tattered photograph to her chest. The storm had taken everything
     , yet here she stood at the cliff's edge, eyes blazing with
     determination. 'This tempest can't erode my spirit,' she thought.
      'For in my heart, the flame of hope burns eternal.'",
  "entry_1": "Through the symbolism of the storm",
  "entry_2": "By illustrating Eleanor's emotional resilience",
  "entry_3": "By depicting the loss of material possessions",
  "entry_4": "All of the above",
  "correct_entry": "entry_2",
  "concepts": ["Reading Comprehension", "Literary Techniques", "Theme
     Analysis"]
}

{
  "subject": 2,  // Literature
  "type": 1,     // Multiple Choice
```

```
  "data": "In the passage, how does the author convey the theme of
     perseverance against adversity?",
  "reading_passage": "Amidst the roaring tempest, Eleanor clutched the
      tattered photograph to her chest. The storm had taken everything
     , yet here she stood at the cliff's edge, eyes blazing with
     determination. 'This tempest can't erode my spirit,' she thought.
      'For in my heart, the flame of hope burns eternal.'",
  "entry_1": "Through the symbolism of the storm",
  "entry_2": "By illustrating Eleanor's emotional resilience",
  "entry_3": "By depicting the loss of material possessions",
  "entry_4": "All of the above",
  "correct_entry": "entry_2",
  "concepts": ["Reading Comprehension", "Literary Techniques", "Theme
     Analysis"]
}
```


The JSON examples illustrate the diversity and complexity inherent in question creation. Each subject and type combination necessitates specific data fields, from LaTeX code for mathematical questions to test scripts for coding queries. This structured approach is essential for creating targeted, effective educational experiences.

We incorporate concepts from the ConceptDatabase to enhance the depth and relevance of questions. For instance, a math question on derivatives might include references to related concepts like functions and polynomials, facilitating a holistic understanding. The structuring of questions aligns with the cognitive tasks discussed earlier: planning (structuring the question's content and format), translation (converting the plan into a structured JSON object), and discrimination (ensuring the question aligns with the learning objectives and student's current understanding). This integration of cognitive processes into question creation exemplifies the application of advanced AI techniques in educational settings, highlighting the potential of LLMs in creating dynamic, context-aware learning experiences.

3.2 Formalizing Cognitive Processes

A cognitive process within this framework is conceptualized as an emulation of human cognitive abilities, which are capable of complex tasks such as teaching and developing nuanced concepts. Language is instrumental to this process, as Dennet Dennett [2015] suggests, because it provides a rich medium for interaction and comprehension between the system in which all humans operate underDennett [2015].

Within the Teach-A-Bull environment, This model is implemented by breaking down educational challenges into distinct, manageable parts that can be handled by a large language model. This approach results in a state-based system that dynamically evolves with each interaction, informed by the continual learning that occurs as the LLM processes the diverse inputs and outputs during its engagement with learners.

Referencing Figure 5, one can conceptualize this methodology as a cognitive planner—a strategic framework for problem-solving via the integration of cognitive tasks. It is within this construct that individual cognitive tasks are interlaced to create a coherent process, with each stage building upon the output of the previous, to advance towards a comprehensive solution.

The potential of this model is especially pronounced in text-centric environments where the manipulation of language is central to the LLM's functionality. Such manipulation allows the LLM to act as a conduit between the objective data-driven world and the subjective experiential realm

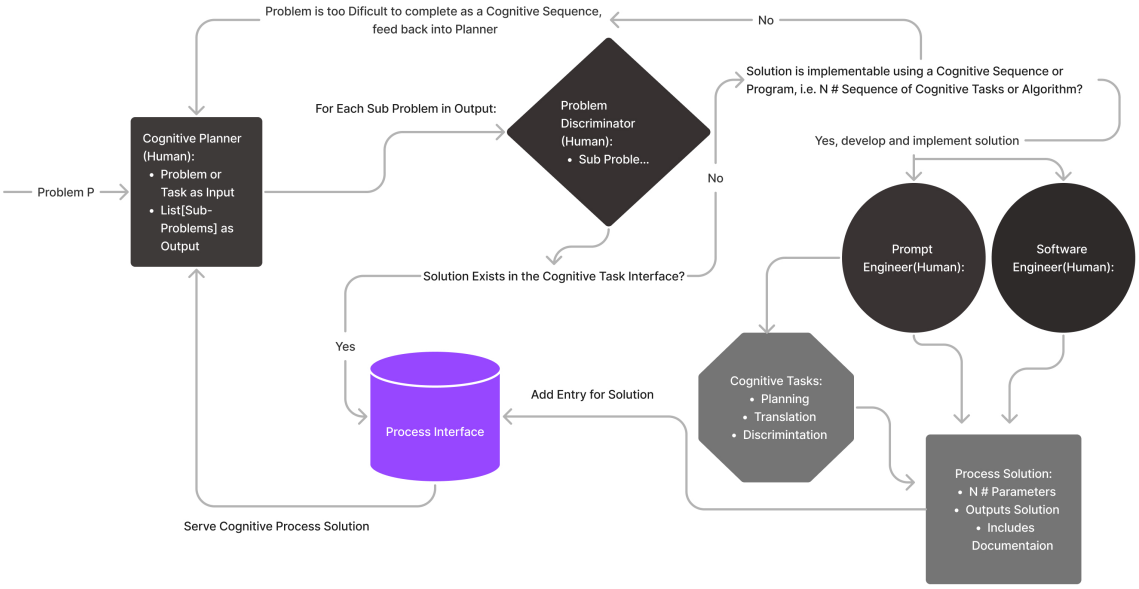

Figure 5: Cognitive Planner: A depiction of the methodological approach to problem-solving through cognitive process integration.

of human interaction.

3.3 Document and Data Generation

Slide Generation The *Generate Slide* algorithm encapsulates these cognitive processes. It begins by initializing the environment and creating a list of slide plans. The algorithm then iterates through these plans, using the planning module to determine the slide's content, translating this plan into a structured object, and adding it to the slide plans list. The discrimination step checks if the current state is optimal for termination. Finally, for each slide plan, the algorithm generates the slide's context, content, and presentation, culminating in a complete slide object. This method exemplifies the integration of cognitive tasks into a cohesive process for generating educational content. In the pseudocode below, s_t contains the Notebank State, the Current SlidePlans, and also the Explored Concepts.

Question Generation The *Generate Question* algorithm follows a similar process. We allow the user to specify the number of questions they wish to answer ahead of time, and also allow them to provide insight into what kinds of questions they would like to answer. Based on these conditions, the generation is a simple iteration generating the number of questions the user asks for based on the conditioning of the environment. This process involves a planning step and a translation step. The tutor plans the content of the question, including the type, subject, and additional required content, based on the condition of the environment. The tutor then translates the plan into a question JSON object which we can execute and utilize in the environment. Our question generation algorithm builds on the simplistic nature of the slide generation algorithm .

Algorithm 1 Generate Slides

Initialize Environment to Generation State
Initialize SlidePlans to Empty List
while not Termination State **do**
    $s_t \leftarrow$ Environment.State
    $plan_t \leftarrow$ Plan Module(s_t) i.e. slide's Title, Purpose, Concepts
    $SlidePlan_{Obj} \leftarrow$ Slide Plan Translation Module($plan_t$)
    $SlidePlans \leftarrow SlidePlans + SlidePlan_{Obj}$
    Termination State $\leftarrow$ Discriminate if s_t is a termination state
end while
Initialize Slides to Empty List
for each Slide Plan SP in Slide Plan Set **do**
    $plan_t \leftarrow$ Generate Slide's Context from SP
    $Content_t \leftarrow$ Parse and translate (Context and $plan_t$) into Slide's Content JSON Object
    $Presentation_t \leftarrow$ Parse and translate (Context, $plan_t$, and $Content_t$) into Slide's Presentation JSON Object
    $Slide_{Obj} \leftarrow$ Slide($SP, Content_t, Presentation_t$)
    $Slides \leftarrow Slides + Slide_{Obj}$
end for

Algorithm 2 Learn Student Session Goal

Initialize Environment to Prompting State
Initialize Query $\leftarrow$ "AI Tutor: How can I help you learn today?"
while not Terminated **do**
    $q \leftarrow$ Student Response to Query
    Environment.ChatHistory $\leftarrow$ Environment.ChatHistory + q
    $s_t \leftarrow$ Environment.State
    $plan_t \leftarrow$ Execute Planning Module(q, s_t)
    Memory Object, Terminated $\leftarrow$ Execute Memory Module($plan_t, s_t$)
    Update Memory Module using Memory Object
    if not Terminated **then**
        Query, Terminated $\leftarrow$ Execute Agent Interaction Module($plan_t, s_t$)
    end if
end while

Algorithm 3 Generate Question Suite

Initialize Environment to Generation State
Initialize Questions to Empty List
for i=1..Number0fQuestions **do**
    $s_t \leftarrow$ Environment.State
    $plan_t \leftarrow$ Plan Module(s_t) i.e. questions' Type, Subject, Content
    $Question_{Obj} \leftarrow$ Question Plan Translation Module($plan_t$)
    $Questions \leftarrow Questions + Question_{Obj}$
end for

Concept Graph Generation The *Generate Concept* algorithm implements a Dept-First Search approach to generating concepts. Essentially, we will start the Graph Generation process by executing generate concept on the **Main Concept**. This will inherently start by generating our first depth of related ideas to the main concept. Our Concept Object being translated contains parameterized definition string containing the $< Concept > ... < /Concept >$ tags, specifying which concepts to map to in the definition. By injecting these tags, the LLM is creating connections to other Concepts which we can parse during the Concept Parsing phase; whether they are in the Graph or not in the Graph yet. If a concept is in the Graph, then we simply map to it. If it is not, we recursively call the generation function on that Concept and repeat the Concept Generation process. The caveat to this is the **depth control** variable, a hyper-parameter in the Concept Generation phase implemented to limit the extension of the graph such that it does not deviate too far from the Main Concept. The concept parsing method implements pattern matching (regex) to parse and extract tokens from the definition the LLM creates for a concept; additionally, we tokenize the definition of a Concept into a List of type $str \cup Concept$.

Algorithm 4 Generate Concept

```
function GENERATE_CONCEPT(concept_name, max_depth)
    Define function ESCAPE_BACKSLASHES_IN_QUOTES(text)
    if max_depth = 0 then
        return Call to LLM API
    end if
    error_msg ← "No current error detected in the parsing system."
    while True do
        llm_output ← Request concept data from LLM
        Perform regex matching and extraction on llm_output
        Assert correct parsing and extraction of YAML data
        Extract and validate concept details from YAML data
        CREATE_FROM_CONCEPT_STRING_ADD_TO_DATABASE(data extracted from the YAML object)
        if max_depth = 0 then
            break
        end if
        Assert successful concept creation
    end while
    return new_concept
end function
```

Translation Error Error during the translation phase occurs frequently enough to need to mitigate the affect it may propagate. Translation must be Type Safe, as Python is Type Safe and executing on objects in Python is therefore inherently type safe. This implies we must incorporate a type checker in our translation modules, and raise $TypeErrors$ when our translation executes incorrectly. It is important to understand that an LLM is simply next token prediction, and can occasionally get formatting or prediction incorrectly due to the probabilistic nature of the generation. Similar to Voyager, we catch execution errors and propagate them into the environment's state as an observation Fan and Anandkumar [2023]. The benefit to this is occasionally the LLM, mainly GPT-3.5-Turbo, will fail to translate correctly. Under these scenarios, our type checker is designed to catch this error and propagate this back into the environment. On an error, we simply

Algorithm 5 Create from Concept String and Add to Database

```
1: function CREATE_FROM_CONCEPT_STRING_ADD_TO_DATABASE(concept_name,
   definition_str, latex_code, ConceptDatabase, curr_max_depth)
2:     if curr_max_depth < 0 then
3:         return None
4:     end if
5:     Initialize def_sequence as an empty list
6:     Create a new concept with concept_name and latex_code
7:     Append the new concept to ConceptDatabase
8:     Tokenize definition_str into tokens
9:     for each token in tokens do
10:        if token contains a concept then
11:            Extract c_name from token
12:            Check if c_name exists in ConceptDatabase
13:            if not existing then
14:                Generate new concept with reduced depth
15:            end if
16:            Append concept or token to def_sequence
17:        else
18:            Append token to def_sequence
19:        end if
20:    end for
21:    Update definition of the concept with def_sequence
22:    return the updated concept
23: end function
```

propagate the error as an observation and re-attempt translation. We will analyze the translation error of GPT-3.5-Turbo vs. GPT-4 on average, since some generation is more likely to fail due to complexity, and understanding how each model performs on the collection of tasks will better help us understand how the system performs as a whole.

4 Process and Results

4.1 Metrics Methodology

In this study, we apply a series of metrics to evaluate the performance of Large Language Models (LLMs) in text-based educational environments:

1. **Knowledge Graph Similarity Analysis:**
    - Collect "Ground Truth" data from field experts.
    - Convert both "Ground Truth" and LLM outputs into vector embeddings.
    - Measure cosine similarity between these embeddings.
2. **Slides/Presentation Similarity Analysis:**
    - Collect "Ground Truth" data from field experts.
    - Convert both "Ground Truth" and LLM outputs into vector embeddings.
    - Measure cosine similarity between these embeddings.
3. **Question Generation Similarity Analysis:**
    - Collect "Ground Truth" data from field experts.
    - Convert both "Ground Truth" and LLM outputs into vector embeddings.
    - Measure cosine similarity between these embeddings.
4. **Actor Generation Capability Analysis (GPT-4 vs. GPT-3.5 Turbo):**
    - Translation Error: Track Number of API requests per error to assess and understand the model's ability as an actor.
    - Information Recall: We evaluate Number of Relevant Concepts Discovered over Number of Discovered Concepts to assess the models understanding of the task.
    - Information Accuracy: We evaluate Number of Factually Accurate Concepts over Number of Concepts to assess the models ability to accurately generate knowledge.
    - Slide Quality: We grade the Discovered Slide Documents on the following scale out of 5 points: ***(1 point)*** *Slide relevancy to the Main Concept,* ***(2 points)*** *If Informative or Exploratory: Slide imposes deep understanding of the Concept/s, Else If Slide Relative: Slide Proposes a meaningful relation between Concepts, Else If Slide Explanative: Slide demonstrates meaningful explanation of challenging material, Else (Examplative) Slide presents a meaningful Example for the Student,* ***(2 points)*** *number of Facts proposed that are true on the slide divided by the number of Facts proposed..* We also add an additional 5 point scale for a slide deque: ***(3 Points)*** *Slide Deque captures the entire Concept Graph,* ***(2 points)*** *Slide presents material in a cohesive and understandable ordering.* We then average the scores together for the Slide Documents and compare.

- Question Quality: We grade the Discovered Examination Questions on the following scale out of 5 points: ***(2 points)*** *Question relevancy to the Main Concept ,* ***(2 points)*** *If Single Concept: question discerns complete understanding of the Concept, Else: Question adequately examines the student's ability to relate Concepts together in a meaningful way.* ***(1 point) if the content is non-repetitive based on the conditions of the environment***, We then average the scores together for the Question Documents and compare.

Similarity Comparison with Expert Data In our research, the evaluation of document generation capabilities is anchored in a similarity comparison to an expert dataset. This methodology is designed to assess the system's proficiency against a baseline established by experts, as well as its performance relative to OpenAI's ChatGPT. We employ cosine similarity of embedding vectors, given that the embedded content encompasses information pertinent to our educational material. We use OpenAI's **text-embedding-ada-002** for embedding vectors due to the large context size and the high dimensionality of the output vectors. This approach involves normalizing documents into a uniform format to ensure that the embedded content remains the primary variable in similarity assessment. Furthermore, we conduct an in-depth analysis of slide generations from different underlying models, namely GPT-4 and GPT-3.5-turbo. While other models like LLaMa 70B were considered, our focus remains on these specific OpenAI models due to time constraints.

Measuring Actor Generation Our method for Actor Generation acknowledges the claim that we previously laid forth prior, which is that text-based environments will serve benefit to document generation and information quality. To evaluate LLM's abilities to generate documents, we propose document-focused evaluations for each document type related to our environment. We use this to understand our Actor's ability to create and develop documents based on the provided context. This also lets us compare and contrast both GPT-4's capabilities and GPT-3.5-Turbo's capabilities respectively, which will provide further insight into the undisclosed training which both models may have undergone.

Note: For better results we could include expert data as a baseline.

4.2 Data Results

Table 1: Concept Generation Comparison on Expert Data

Subject	Economics	Calculus	Data Structures	NLP
LLMaiT-BE	0.9926427155	0.9964194525	0.987759412	0.9905676651
GPT-4	0.9880887679	0.9946124417	0.9817208341	0.9956680059

Table 2: Slide Generation Comparison on Expert Data

Subject	Economics	Calculus	Data Structures	NLP
LLMaiT-BE	0.9643093751	0.9458801326	0.9760217327	0.9696427139
GPT-4	0.9787711068	0.9498781766	0.9780565495	0.9787162149

Table 3: Question Generation Comparison on Expert Data

Subject	Economics	Calculus	Data Structures	NLP
LLMaiT-BE	0.9474690584	0.9312320928	0.9088540586	0.8710491658
GPT-4	0.966644659	0.9246734959	0.9155144506	0.9156977609

Table 4: Agent Ability Metrics

Actor	Translation Error ($Reqs/Error$)	Information Recall ($Rel/Discov$)	Information Accuracy ($Acc/Discov$)	Slide Quality	Question Quality
GPT-4	11.95245098	0.939883325	0.9873728198	8.4	7.25
GPT-3.5-Turbo	4.243444056	0.7926136364	0.7451298701	4.075	3.05

5 Discussion

5.1 Similarity Ratings

Regarding our methodologies, we will reflect on the utility of our process. Asides from exceeding rate limits for massive amounts of generation requests, the similarity between all of the documents did not offer any distinguishing results for any of the topics. First, the similarity rating we used two compare GPT and our LLM Agent were not necessarily reflective of quality. Simply, the similarity rating from the embeddings reflected the general idea of a document and did not offer more information from here. On average, our two measured values were no more than 0.05/1.0 distance from . Reflecting on why this may be may bring further insight into the quality of this as a metric. The embeddings are simply a vector used to predict the next token. Based on the OpenAI *text-embedding-ada-002* which was never at any point specifically trained to measure quality of response. Some may contemplate whether or not an embedding vector may contain a quality metric in the high dimensionality encapsulates this, however looking further into this topic has shown that more research would need to be done in order to determine this. We will mention however that a reward based model can potentially be used outside of our similarity analysis to gauge which document is preferable for generation. We attempted to use this as an addition to the similarity analysis, however from information we found the OpenAI embeddings model was not used on reward modeling for RLHF, so the information regarding quality which may be incorporated into the model if it were a reward-based model is not necessarily incorporated into our findings.

5.2 Actor Ability Metrics

From an Actor stand point, we evaluate GPT-4 vs. GPT-3.5 in terms of performance in the Tutor Environment. Our findings demonstrate what many would expect; GPT-4 performs better in all categories than GPT-3.5. In retrospect this may be a result of the training process, which is important to understand the primary goal of the GPT-3.5-turbo assistant vs. the GPT-4 assistant. During the Assistant fine tuning process, GPT-3.5-Turbo was trained prior to the emergence of the Assistants and tool-using GPT-4 models. Essentially, GPT-4 is not only the bigger and more

powerful model, but it has also been finetuned on generalized Agent tasks, making it more adept to acting in our unknown environment. We can conclude that GPT-3-175B has primarily been fine tuned on Assistant tasks which make it respond in a assistant-like manner, making it harder for GPT-3 to generalize to other tasks **?**. Although the training process for GPT-4 has been largely closed source, we can discern that a lot more effort has been placed into generalizing the Agent capabilities of GPT-4, which our findings would also support. To discuss GPT-3.5-turbo, many times it seems to be confused, as in the generations are just reiterations of the engineered prompt; which is not an issue for planning, however translation of the engineered prompt into a Slides Object is not intended and does not serve any purpose towards constructing a quality education document. Additionally, GPT-3.5-turbo did even worse at generating questions, where it generated coding questions for an economic student5.2, a mistake which no real human would make given the context. From a generation standpoint, GPT-3.5-turbo is not a feasible option for many of the tasks we were performing, since our goal with the document generation was quality over ChatGPT. To finalize our conclusions on GPT-3.5-turbo, it performs adequately on most tasks about half of the time, where the other half it fails extremely bad as if it were confused. We bring this up to reiterate the phenomenon of how an LLM will occasionally "*hallucinate*" ideas which do not exist into the context into the context, or more noticeably, do not understand how to function in our text-based environment.

Example 5. GPT-4 is able to define code test cases with accurate numerical value assertions

```
{
    "subject": 1,
    "type": 3,
    "data": "Write a Python function to calculate the Edit Distance
       between two strings using the Levenshtein distance algorithm.
       Your function should take two arguments (string1, string2) and
       return the minimum number of operations required to convert
       string1 into string2. You can limit your solution to insertions
       , deletions, and substitutions as the allowed edit operations
       .",
    "boilerplate": "def edit_distance(str1, str2):\n    # Your code
       here\n    pass",
    "test_cases_script": "assert edit_distance('kitten', 'sitting') ==
        3\nassert edit_distance('intention', 'execution') == 5\nassert
        edit_distance('algorithm', 'altruistic') == 6",
    "concepts": [
        "Edit Distance",
        "Applications of Edit Distance Algorithm in NLP"
    ]
}
```

Example 6. GPT-4 fails to demonstrate adequate actor generation

```
 {
    "subject": 1,
    "type": 3,
    "data": "Imagine you are implementing a graph data structure to
       represent a simplified map of a network. Define a 'Graph' class
        that uses an adjacency list to store nodes and edges.
```

```
        Implement methods for adding nodes (add_node) and edges (
        add_edge), and a method for performing Breadth-First Search (
        BFS) from a given start node. Demonstrate your implementation
        by creating a small graph and performing a BFS.",
    "boilerplate": "class Graph:\n\tdef __init__(self):\n\t\t# Your
        code here\n\n\tdef add_node(self, value):\n\t\t# Your code here
        \n\n\tdef add_edge(self, from_node, to_node):\n\t\t# Your code
        here\n\n\tdef bfs(self, start_node):\n\t\t# Your code here\n\n#
         Example usage\n# Create a graph instance\n# Add nodes and
        edges\n# Perform BFS",
    "test_cases_script": "def test_bfs(graph):\n\t# Your test cases
        here\n\tassert graph.bfs(start_node) == expected_output,\n\t'
        Failed test: BFS did not find the nodes in the expected order
        '",
    "concepts": [
        "Graph Data Structure",
        "nodes",
        "edges",
        "adjacency list",
        "Graph Traversal Algorithms"
    ]
}
```


Example 7. GPT-4 mistakes a coding question for a free response question


```
{
    "subject": 1,
    "type": 0,
    "data": "Implement a function `count_vowels` in Python that takes
        a string input and returns the number of vowels in the string.
        Consider the following vowels: a, e, i, o, u, and their
        uppercase counterparts.",
    "rubric": "Rubric: [1 Point] Correct implementation of the
        function; [1 Point] Correct count of vowels; [1 Point] Correct
        handling of uppercase vowels; [1 Point] Clean, readable code
        with appropriate comments.",
    "concepts": [
        "Regular Expressions"
    ]

}
```


Example 8. GPT-3.5-turbo asks an economics student a coding question


```
{
    "subject": 1,
    "type": 3,
    "data": "Write a Python class `StateEconomy` to simulate State
        Centered economics. The class should include methods that
        somehow reflect governmental intervention in markets. Implement
         methods `intervene_market` and `calculate_gdp`, where `
```

```
        intervene_market` alters unspecified economic parameters, and `
        calculate_gdp` returns a GDP value based on these interventions
        .",
    "boilerplate": "class StateEconomy:\n\tdef __init__(self):\n\t\t#
        Initialize state parameters\n\n\tdef intervene_market(self,
        intervention):\n\t\t# Apply state intervention in market\n\n\
        tdef calculate_gdp(self):\n\t\t# Calculate GDP based on state
        interventions\n\n# Example usage\n# Create StateEconomy
        instance\n# Apply interventions\n# Calculate GDP",
    "test_cases_script": "def test_state_economy(economy):\n\t# Your
        test cases here\n\tpass",
    "concepts": [
        "GDP"
    ]
}
```

One specifically interesting case was during Question Generation, where GPT-4 is correctly able to assess and develop proper test cases for coding questions 5.2. The process in which to determine this requires execution of correct code to validate the test cases. Although we were able to observe proper assertions in many cases, this does not mean that GPT-4 was completely errorless. In various cases, it would fail to output the correct assertions, fail to understand the proper requirements, and occasionally even fail to generate the question types with the correct parameters, or even include parameters which did not exist. More common than the ladder, we can view a correct translation, i.e. the object is parsable and valid for program execution, however there exists holes in what it is able to produce. In Example 75.2, we can see that while although the Agent intended on creating a free response question, the question prompt describes a coding question. Now although these errors are few, I would make the case that this one of the most interesting discoveries, as we may be able to say that although it intended on having the user generate code, it is an error that no human would make in a question design process. To reflect why it may have made such an error is hard, since it reflects poor execution in both planning and translation. I would speculate, however, that in this state the model was confused about the intention of questions used for programming, thus it has confused free response with code.

Slide generation via the Agent was the best performing generation out of the two. This may seem weary as to why this may be, however we will provide further detail into our generative process which may explain why this may be the case. The amount of computation we invest in generating slides is almost double of which we invest in questions. In fact, we have simply a single planning process which feeds into the translation, meaning if something goes wrong during the planning it would be very hard for the Agent to recover from this error. With slides, however, we invest a whole phase of planning prior to even generating the content, essentially we abstract out a plan as representing the document/data structure which we then iterate on through another process of planning and translation. In other words, we construct an overall plan for the structure of the document as a whole prior to narrowing the focus into each object selectively.

5.3 Cognitive Process Design

Reflecting on the process design reveals insightful and promising avenues for continued research. To reiterate how the generation process works; *(for Concepts)* we construct an unstructured plan

of content, identify the central focus, and recursively discover related concepts via a depth-first search, *(for Slides)* We implement a plan for the data structure, and then execute on that plan resulting into a translated copy of the final data structure, *(for Questions* we sequentially plan and translate for each question into the final data structure.

Speed The Concept process takes relatively longer than all other processes due to the recursive and exploratory nature of its operation. The justification for this originally was for emulating how humans may interpret information, where we can explore different contexts of terms and what they represent. This did not play well into the speed of Concept Generation. The algorithm for Concepts runs in $O(b^d)$ time, where b is a branching factor and the depth is a hyperparemeter which we chose to be 4. The branching factor is an interesting notion in this case, since theoretically the branching factor can be equal to the size of the context, or at least the number of possible token sequences which represent a branch, i.e. *¡Concept¿ name of concept ¡/Concept*, which is how the Agent creates branches to new locations. I would propose however this is unlikely, and that an estimated number could be proposed as an average case. We did not measure the average value, since this would require an extensive number of topics and an exorbitant number of tokens, which for us falls far outside of our budget for this project. Less scientifically, after running numerous generations we obtained 15 Concepts for the lowest generation with GPT-4 and 86 Concepts for the largest generation, giving us, assuming no re-referencing to already discovered nodes, a range of 2.11 and 3.04 for an appropriate average. We obtain these values by taking the $d - th$ root of the min and max, since these would represent the assumptions we previously made.

The question generation was also slow, this was due to the extensive planning and computation put into the sequential format of the questions process. The slides process was relatively quick due to the parallelization possible once you have the overall structure of the data structure. In other words, once you have a high-level plan for the data structure as a whole, you can iterate on each piece in parallel. This functions as an extremely powerful performance increase, since if generating a slide takes N seconds, sequential operation for M slides would take $O(M * N)$ time, whereas for us it would take approximately $O(N) + O(1)$ seconds instead, where the additional time is overhead in managing thread processes. This idea can even be taken one step further by adjusting the time taken to plan, since we currently take $O(M * N)$ time to plan the slides, generating the plan is the expensive part. To extend this notion of planning, one could further research whether or not the sequential nature of planning is beneficial, which it may not be. We simply format the planning process as a sequential algorithm by arguing that a Markov Model can accurately reflect how a human would process this, i.e. you plan what the first slide has, then you observe and plan the second, observe and the third, and so on. This was also under the assumption that NTP could not accurately simulate a Markov Process capable of doing the same through its own *Next Token Prediction*, and these assumptions are grounds for further research.

Utility The utility of the model we proposed is hard to definitively answer without more research, since we were unable to demonstrate any conclusive measurement of improvement over the ChatGPT model. We acknowledge many of our findings also suggest that the proposed architecture for our model would need further improvement and iteration in order to conclusively determine its effectiveness. With that being stated, one can speculate that the architecture serves at least enough utility to pragmatically generate documents at a scale which would outperform ChatGPT, since we are not inherently limited by a context window. We can also observe that our proposed model can aid in the forget-full nature of the base model, since the memory module provides long-term working memory, similar to that proposed in the VoyagerFan and Anandkumar

[2023] paper as well as the Generative Agents Park et al. [2023] study conducted. One can also infer that being able to work and operate on documents at different levels of abstraction could offer beneficial results given the problem structure is complicated enough to support the reasoning behind doing so. In order to justify this proposal, we consider the application which more research could provide further insight on, where instead of a Learning Environment we simulate a Developer Environment, where the action space is executable programs as opposed to slide deques and exam questions. In this environment, one can infer the requirement for abstraction, since programmers deal with abstractions at every level of the development stack.

5.4 Final Reflection

The primary constraint in our approach is the human engineer. This is not to undermine their role in developing the process, but rather to highlight our inherent limitations in effectively utilizing and applying tools like Large Language Models (LLMs) in this domain. At every stage, humans must decide which sub-processes are necessary for task completion. For straightforward tasks, this decision-making is quite simple. However, as tasks grow in complexity, it demands deeper introspection from the engineer on how they would approach problem-solving, a process that isn't always transparent. This challenge is amplified by Cognitive Introspection, where engineers must scrutinize their own problem-solving methods, some of which they might not be fully conscious of. As a result, determining a definitive strategy for navigating complex problem-solving tasks becomes difficult. Hence, the approach of breaking down a problem into smaller, manageable sub-problems remains a challenge yet to be fully resolved.

6 Conclusion

Synthesis of Findings

Our investigation into Large Language Models (LLMs) as actors in text-based environments yielded insights into their document generation capabilities and the cognitive processes underlying their interactions. While GPT-4 demonstrated superior performance across multiple metrics, our comparison revealed that similarity ratings were not definitive indicators of document quality. The slide generation process outperformed question generation, likely due to the pre-planning of a higher level document structure that facilitated parallel content creation at smaller scales. Further investigation and research into developing a model primarily based in this idea would prove to result better analysis and lead to more definitive conclusions about the utility of our model.

Considerations and Reflections

The utility of our model, though not conclusively superior to ChatGPT, suggests potential in scaling document generation beyond context window limitations. The architecture's design allows for extended working memory and varied levels of abstraction, which could be beneficial in more intricate environments like software development. The human engineer's role emerged as a pivotal yet constraining factor, with our cognitive introspection highlighting the challenges in devising strategies for complex problem-solving. Further research is warranted to refine the approach and fully harness LLMs' capabilities.

Future Directions

Looking forward, the focus should be on enhancing the model's architecture for greater efficacy, exploring its application in other domains, and reducing the overhead of human decision-making for sub-process identification by providing formalization into problem construction. The exploration of LLMs in alternative environments requiring more abstract operations on data may also be an important direction to further engage in.
In summary, our inquiry into the capabilities of Large Language Models has illuminated the intricate dance between emergent intelligence and language. This exploration has not only expanded our grasp of LLMs as dynamic agents but also underscored the complexities inherent in coupling cognitive processes with effective problem-solving. We have uncovered the necessity of deconstructing a grand challenge into solvable fragments, a task fraught with assumptions that must be meticulously examined. Our findings gesture towards an intriguing hypothesis echoed by DennettDennett [2015]: the mastery of language may well be the key to unlocking intelligence. Yet, the true essence of this does not lie in only understanding language, but also understanding in how to use language to discover the answers that we seek.

References

Daniel C. Dennett. The role of language in intelligence. Tufts Archival Research Center, 2015. URL `https://dl.tufts.edu/concern/pdfs/3r075579n`. This article was reprinted in Germany, but not translated into German.

Sifatkaur Dhingra, Manmeet Singh, Vaisakh SB, Neetiraj Malviya, and Sukhpal Singh Gill. Mind meets machine: Unravelling gpt-4's cognitive psychology, 2023.

Linxi Fan and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

A. Kolmogorov. Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, 14(5):662–664, 1968. doi: 10.1109/TIT.1968.1054210.

A. A. Markov. *Theory of Algorithms*. 1954. [Translated by Jacques J. Schorr-Kon and PST staff] Imprint Moscow, Academy of Sciences of the USSR, 1954 [Jerusalem, Israel Program for Scientific Translations, 1961; available from Office of Technical Services, United States Department of Commerce] Added t.p. in Russian Translation of Works of the Mathematical Institute, Academy of Sciences of the USSR, v. 42. Original title: Teoriya algorifmov. [QA248.M2943 Dartmouth College library. U.S. Dept. of Commerce, Office of Technical Services, number OTS 60-51085.].

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

Alex Pillen and Emma-Kate Matthews. Natural language modelled and printed in 3d: a multidisciplinary approach. *Humanities and Social Sciences Communications*, 9(1):72, 2022. doi: 10.1057/s41599-022-01089-5. URL `https://doi.org/10.1057/s41599-022-01089-5`.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment, 2017.

Ilya Sutskever. Large language models and transformers. Workshop Presentation at Calvin Lab Auditorium, August 2023. Available: https://simons.berkeley.edu/talks/ilya-sutskever-openai-2023-08-14.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, Ji-Rong Wen, Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*, 2023.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.