Predictive Idea Generation Through Clustering Narrow Word Embeddings

Jackson Grove
College of Information Science, University of Arizona
ISTA 421: Introduction to Machine Learning
Salena Torres Ashton
December 12, 2024

Problem & Significance

Large language models (LLMs) have transformed software usage and development since 2022 with the release of ChatGPT, the first commercially available LLM. This sparked a revolution that would become the current consumer AI market – cheap intelligence made easily accessible to consumers, with dozens of companies simultaneously pouring multi-billion dollar investments into the technology's growth. LLMs are now integral to enhancing writing, generating and debugging code, and on larger scales, even powering entire businesses by serving as the backbone for AI agents automating workflows. The rapid and extensive scaling of LLMs is not just driven by their ability to generate text that mimics the semantics of human writing, but more profoundly by their capacity to synthesize and deliver human ideas.

Surprisingly, this capability is not what these models are explicitly engineered to achieve. At their core, LLMs are pre-trained neural networks that iteratively predict the next word in a text sequence, typically based on a user's prompt. By scaling these models massively, we've unlocked an emergent property enabling them to deliver coherent and relevant ideas in response to the user's input. This approach has proved remarkably successful in recent years. Yet, as the demand for more intelligent models grows, a new perspective emerges – what if we shifted focus from generating the next word in a sequence to directly generating the next idea? By clustering words within idea-dense text, we can define ideas more concretely, and leverage this newly defined latent space to develop a new class of generative models – models designed not as next word predictors but as idea generators, a fundamentally different way of building intelligent AI systems.

Data Exploration

A dataset of 10,000 scientific paper abstracts published on ArXiV under the STEM tag were used in building this model. The dataset contained the abstract texts which were later extracted for cleaning and preprocessing. In its entirety, the dataset included the following features: paper ID, submitter, authors, title, comments, journal reference, DOI, report number, ArXiV categorization tags, license, abstract, version numbers and version history, date of the last update and contributing authors.

A dataset of scientific paper abstracts was ideal for this model due to the high density of ideas contained within academic writing. Abstracts often present complex and novel ideas using concise language, providing readers with a clear overview of the research being presented while maintaining brevity. Additionally, many abstracts follow a logical progression of ideas, offering justification for the methods presented. This combination of succinct complexity and logical flow of ideas is well-suited for both capturing a broad range of ideas, each requiring relatively few words to express – and for predicting those ideas in sequence.

The original data was significantly larger than the 10,000 papers used for this project. Due to computational resource limitations, and with Salena's permission, I extracted the first 10,000 papers from the total dataset of approximately 1.7 million papers. It's worth noting that this larger dataset itself is a subset of the complete ArXiV bulk data, which contains millions of additional papers from outside the STEM tag. While this project is limited in scope due to resource constraints, it serves as a foundational proof of concept for the methods shared. With this, it is important to note the abundance of data that would make it easily scalable for the future.

Data Preprocessing

The abstract data was split into both words and string permutations of words, ranging from 1 to 5 in length (with 1 being the word itself). These word permutations were important in capturing the nature of phrases oftentimes representing ideas much more nuanced than the product of the words that make them up. Contrary to many other methods, the stop words were not removed from this dataset as they are important for giving the necessary context to ideas. For example, consider the following two sentences:

"She saw a dog with a collar."
"She saw dog collar."

The first is unprocessed while the second has common stop words ("a" and "with") removed. While these words do not carry much meaning individually, the inclusion or exclusion makes a significant difference in the final interpretation. The first sentence clearly describes someone seeing a dog wearing a collar while the second could imply someone seeing a collar with no mention of a dog. To avoid semantic ambiguity and loss of intended meaning, the data was processed into individual words and short phrases but preserved completely in its contextual form beyond that. The only transformation that was done to the words and phrases themselves was forcing them into lowercase so as to ensure the token can be recognized in any context regardless of differing capitalization.

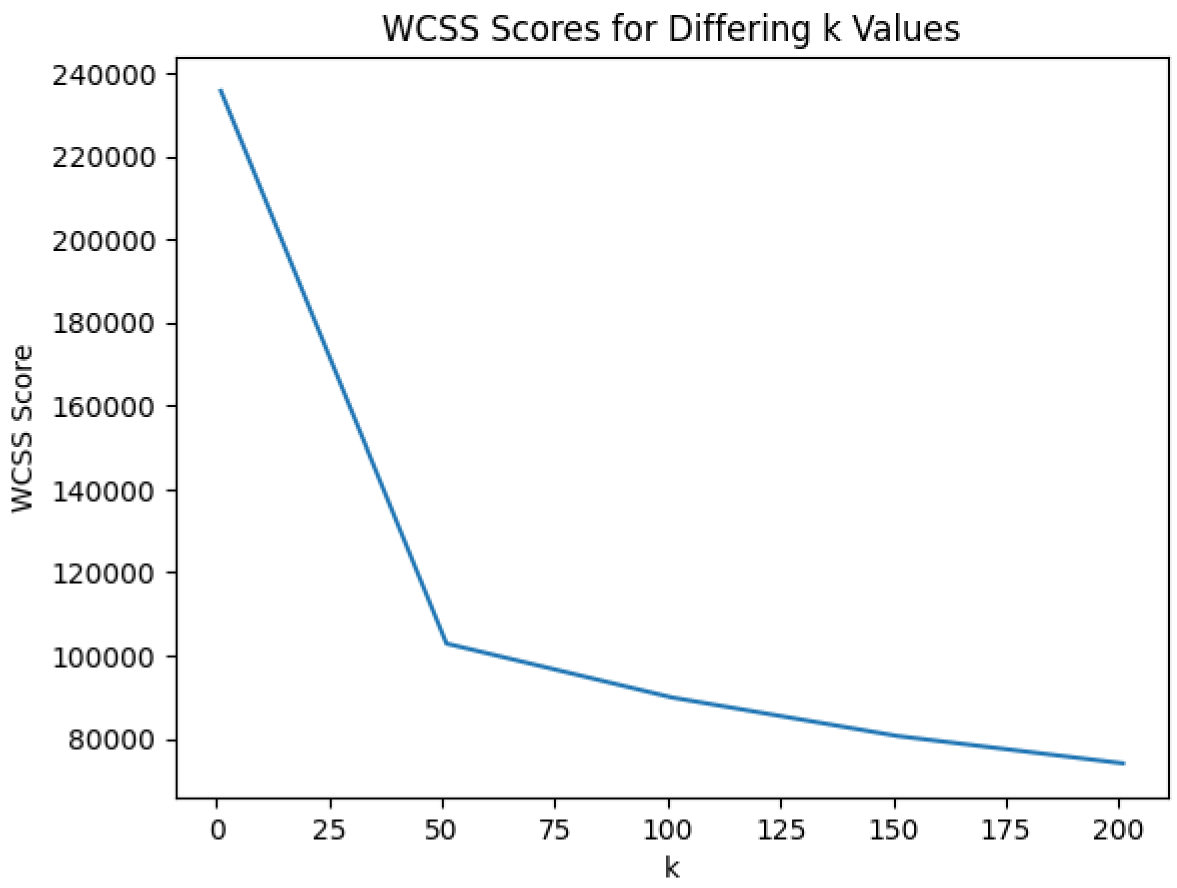
Model Selection, Application & Exploratory Data Analysis

Given the words and short phrases extracted from the dataset, Word2Vec, a popular pre-trained word embedding model available from the Gensim Python library, was employed as an initial step to cluster tokens based on their semantic meanings. The Word2Vec model transformed these words and phrases into 1000-dimensional vector embeddings, positioning tokens with similar meanings closer together in the embedding space, and positioning those with different meanings farther apart. This semantic-based positioning enables the ability to compute distances between points, facilitating the clustering of words and phrases with similar semantics.

For clustering, k-means was selected due to its simplicity, explainability and effectiveness in high-dimensional spaces. The primary goal of clustering was to define one distinct idea per cluster, where each cluster encapsulates words and phrases synonymous with a common concept. For example, the tokens "*caused*", "*driven*", "*produced by*" and "*generated*" might form a cluster representing the notion of causality. The class labels of these clusters were then used to represent and track these specific ideas. The task of defining distinct idea clusters programmatically is valuable in itself, but it also lays the foundation for later stages where a generative model will be built to predict and synthesize such ideas.

The most significant challenge in clustering semantically similar words and phrases, however, is choosing an appropriate *k* (the number of clusters). This decision involves balancing the fidelity of ideas captured with the quantity of data available for training – both of which directly impact the performance of the generative model in the next phase of the pipeline. In any dataset, there are naturally sparse, novel points (ideas in this case) that contribute to the dataset's diversity, alongside more common ones that are less unique and interesting. Setting a low *k* will result in a small number of clusters, each containing many data points, but risks overgeneralizing distinct ideas into broad, common categories. For instance, clustering terms like "artificial intelligence", "number theory", "quantum" and "gear ratio" together under a generic STEM-related cluster would obscure the meaningful differences between these concepts.

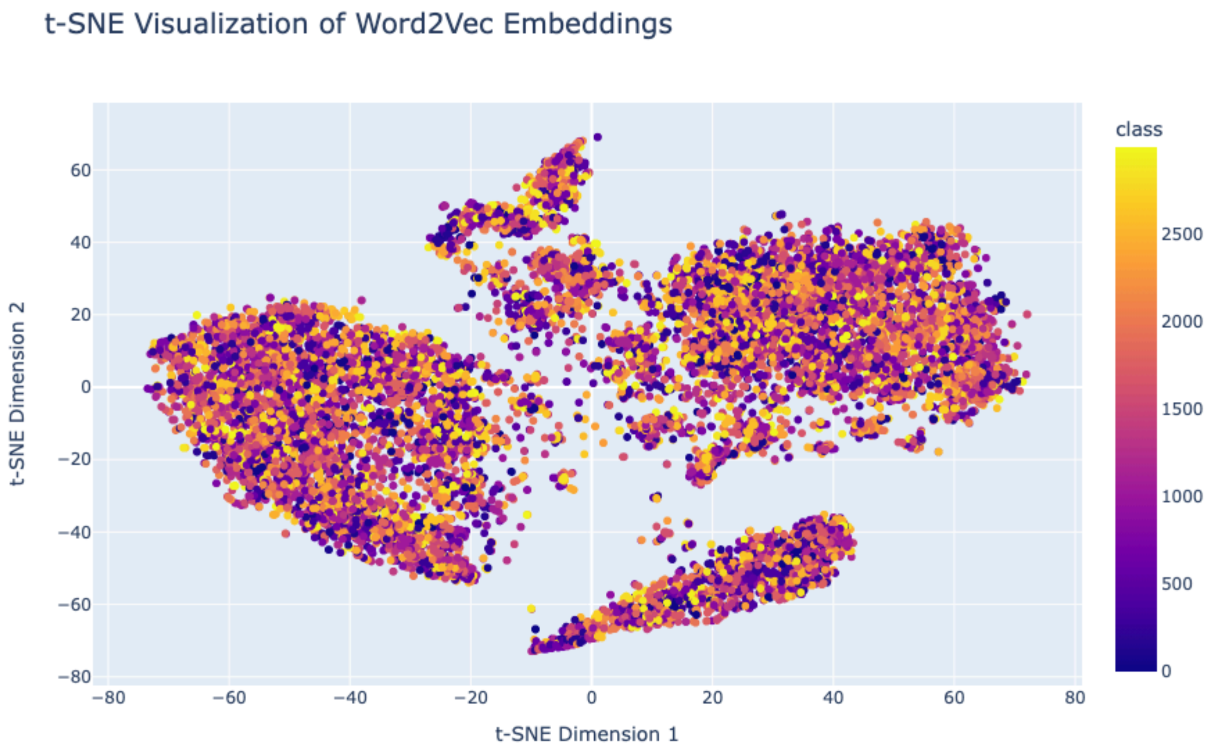
Conversely, setting a high *k* allows for the distinction of more specific and interesting ideas, but runs the risk of creating clusters with too few data points to train a robust model. To find an optimal *k*, the elbow method was used which analyzes the model's WCSS (within-cluster sum of squares) scores to determine the point of diminishing returns as *k* increases. The elbow method identified *k* = 50 as the optimal value, which was surprisingly low given the high expected diversity of ideas in academic writing. In a dataset of 10,000 abstracts and over 11,000 unique word tokens, the idea that there are only 50 distinct ideas within such a vast and diverse volume of data seemed implausible and raised further considerations.



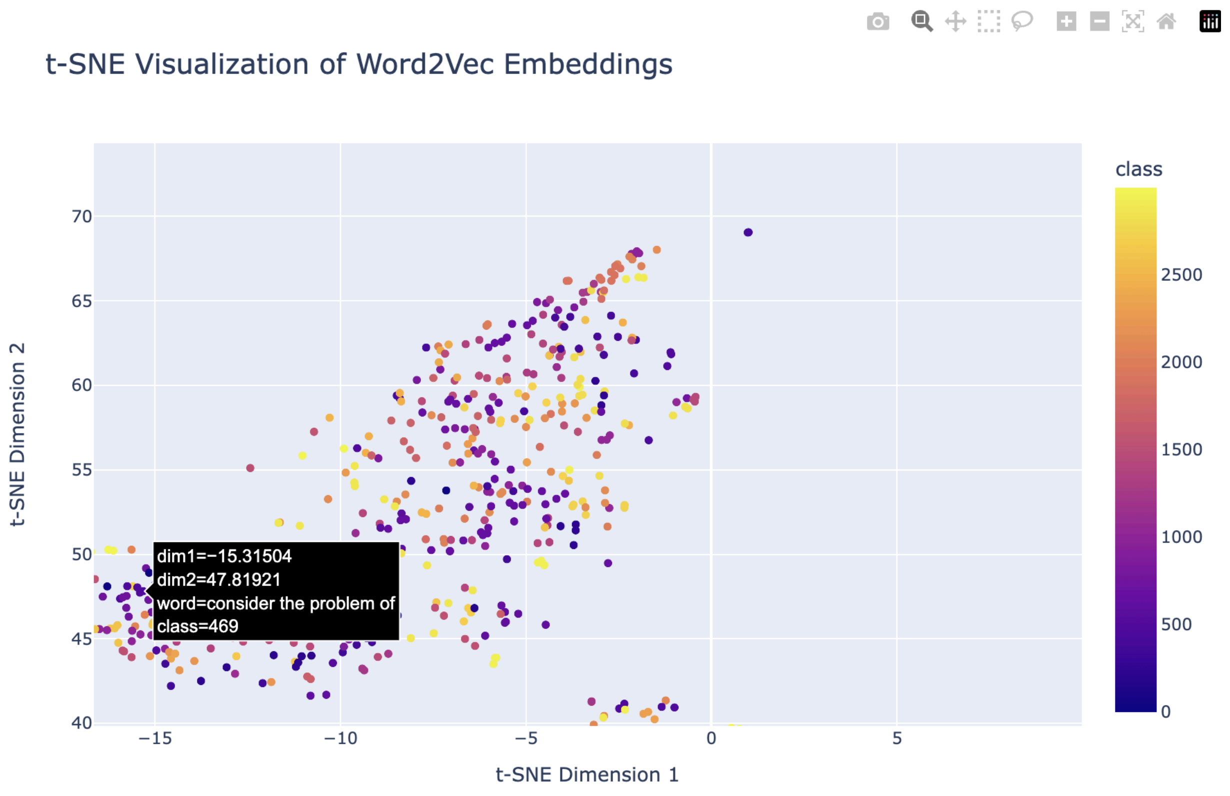
The plotted WCSS graph highlighting an optimal *k* = 50 via the elbow method.

For this project, I opted to prioritize high-fidelity of ideas over reliable access to adequate quantities of data in each cluster, recognizing that this project, as a proof of concept, can be scaled up in the future to incorporate a larger dataset with more computational resources to support its processing. By clustering in favor of specificity, the model emphasizes capturing the diversity of novel ideas, even if that comes at the cost of some sparsely populated clusters. With this reasoning, I initialized the model with an arbitrary *k* = 3000 instead of the proposed *k* = 50 which seemed much more aligned with the true number of ideas present in the dataset. This *k* will later be refined in the evaluation phase.

With the tokens clustered and ideas defined, we reach the stage where exploratory visualization is possible. In order to do so, the 1000-dimensional vector embedding space must be reduced to 2 or 3 dimensions, in which t-SNE was selected to do so. t-SNE, unlike other dimensionality reduction techniques like PCA, performs very well in separating nonlinearities in high dimensions – however since this method is only being used for creating the visualization of clustered embeddings, the performance does not affect anything other than the placement of points in the visualization. Using t-SNE to reduce the clustered embeddings to two dimensions, an interactive visualization was created, plotting word embedding tokens, colored according to their class label clustered by k-means, complete with each points' word, class label, and 2D coordinates upon hover.



2D projection of Word2Vec token embeddings as an interactive visualization with points colored according to k-means class categorization (k = 3000).



Zoomed in view of interactive visualization of clustered Word2Vec embeddings, highlighting metadata visible with the hover functionality.

From exploring the classifications and semantics of neighboring tokens through the visualization, the model demonstrates strong performance at clustering tokens into common ideas. Phrases such as "*consider the problem of*", "*here we show that*", and "*as well as for*" are clustered which makes semantic sense as they function interchangeably as transitions when introducing examples to illustrate a point. These findings signify that our model is performing well and $k = 3000$ is a reasonable estimate for the number of ideas expressed in the dataset. Further tuning and refinement of k will be addressed in the evaluation stage to enhance clustering precision.

Building the Generative Model

With these newly clustered embeddings it is now possible to create a generative model that leverages the sequence of class labels from the clusters – each representing a group of semantically similar words or phrases associated with a common idea. Unlike traditional generative models, which predict individual words or phrases, this approach focuses on predicting the class labels of these clusters. By doing so, the model emphasizes the progression of ideas represented in the text, rather than focusing narrowly on the direct progression of specific words.

This abstraction shifts the task from word-level generation to idea-level generation, where each class label corresponds to a broader semantic concept – referred to as an "idea" from this point forward. For instance, instead of predicting the specific word "*explain*", the model predicts the class label for the cluster associated with it. Each class label represents a cluster containing a list of semantically similar terms, such as:

```
["explain", "illustrate", "clarify", "demonstrate",
"justify", "expand upon", "elaborate"]
```

This results in a sequence of class labels that can then be mapped back to their corresponding clusters for generation. For example, take the following sequence:

```
456, 2, 1945, 30, …
```

Each class label within the sequence maps to their corresponding clusters, containing the following tokens:

```
["the earth", "earth"],
["has", "possesses", "contains"],
["many", "significant amount", "plentiful", "sufficient"],
["lifeforms", "living beings", "organism", "organisms"], …
```

This sequence of class labels encodes for the thought that the Earth has a significant amount of lifeforms. Not only are the terms within these clusters designed to be synonymous with a common idea, but they also provide nuance and a much more complete notion of the idea's meaning. In many cases, this can offer more meaning than the original word can provide. For instance, take the cluster for "*explain*": while the word "*explain*" suggests clarification, its cluster –

["*explain*", "*illustrate*", "*clarify*", "*demonstrate*", "*justify*", "*expand upon*" and "*elaborate*"]

– paints a much richer and more complete picture of its meaning. It represents not only clarification and demonstration as is typically associated with the word "*explain*", but also elaboration and justification to achieve a full *explanation*.

For the generative model, an n-gram approach was chosen for its simplicity and explainability. While more sophisticated models could be employed in the future, this proof of concept prioritizes transparency and explainability over raw performance. The primary focus of this project lies in knowledge representation and exploring new approaches to tokenization, rather than optimizing for a high-performance generative model. This is not to say that these methods cannot lead to highly performant models, but the emphasis here is on how generative models, as extensions of these novel knowledge representation techniques can perform – focusing on explainability over absolute performance.

By first mapping each word or phrase token to its respective class label – and vice-versa to track both word-to-cluster and cluster-to-words pairings as previously discussed – the sequence of class labels can be analyzed to create a frequency distribution for each class label token. To clarify, while words and phrases were previously used as tokens for clustering, the n-gram model requires class labels as tokens. From this point forward, the term "tokens" will refer to the sequence of class labels used as n-gram inputs. These frequency distributions are initialized for each token by transforming/tokenizing the original text into class label tokens and iterating through the sequence to build a list of the next tokens that follow each token. Once the frequencies of each next-token observation are recorded, the model can iteratively generate the next token in a sequence, allowing it to construct strings of ideas. To account for when the sequence should stop, a special stop token, *END,* is introduced as a manufactured class label, representing the end of the original text. After processing, the tokens and their respective frequency distributions are structured as follows when the n-gram hyperparameter $n = 2$:

```
{
  (456, 253):[1, 1, 56, 33],
  (253, 44):[90, 35, 35, 2912, 5, "END"],
  (55, 1163): [88, 6, 7, 7, 7, 903],
  (249, 499): [1988, 5, 443, "END", "END", "END", "END"],
  …
}
```

Each token is expressed as an immutable tuple containing 2 class labels (since $n = 2$). These tokens are mapped to the class labels that appear immediately afterwards in the training data, with the frequency of each next-token observation proportional to its prevalence in the original sequence. As a result, the lengths of the lists vary since not all tokens are equally common in the data.

During generation, the next token is selected either randomly – preserving the probabilities defined by the frequency distribution – or deterministically as the most frequent token, depending on the boolean state of the *temperature_0* parameter. This generation process repeats until either a token is generated that does not exist within the dataset, meaning there are no associated next-tokens for it, or until an *END* token is predicted. The generation function is programmed to take in text as input, which it will then convert to class labels and predict upon, displaying each generated token as its list of words and phrases (seen in examples above).

Evaluation & Results

The evaluation of the model was conducted through both qualitative analysis of the generator's outputs and Cohen's Kappa to assess the quality of the knowledge representation clusters.

The generator produced outputs that were generally coherent and sensical, however the performance varied significantly depending on the choice of *n*. With small *n* ($n = 1$ to $n = 2$) the model often generated repetitive sequences of the same class labels, not having the necessary context to recognize it is cycling through the same few output tokens. With large *n* ($n = 5$ and beyond), the model produced much more coherent and impressive outputs but encountered issues when attempting to generate tokens that did not exist in the training data, leading to errors that stopped the generation process. For most prompts, the model was only capable of generating a few successful token generations. For generic prompts, however, the model was much more capable of generating valid output tokens, providing interesting approaches to research problems – reflective of the scientific and academic writing it was trained on.

The knowledge representation clusters were evaluated with Cohen's Kappa for 6 subsections of the latent representation space. The annotators were tasked with evaluating 100 clusters, analyzing if all points within each group are correctly clustered with none that should be removed. Their agreeance with each cluster was recorded in the table below:

	Annotator 1			
Annotator 2		Agree	Disagree	Total
	Agree	76	4	80
	Disagree	4	16	20
	Total	80	20	100

Using the following formula, the Cohen's Kappa score was calculated:

$$\kappa = \frac{p_0 - p_e}{1 - p_e}$$

Where:

$$p_0 = \frac{Number\ of\ agreements}{Total\ samples}$$

$$p_{Agree} = \frac{Row\ 1\ total \bullet Column\ 1\ total}{N^2}$$

$$p_{Disagree} = \frac{Row\ 2\ total \bullet Column\ 2\ total}{N^2}$$

$$p_e = p_{Agree} + p_{Disagree}$$

We calculate:

$$p_0 = \frac{76 + 16}{100} = 0.92$$

$$p_{Agree} = \frac{80 \bullet 80}{100^2} = 0.64$$

$$p_{Disagree} = \frac{20 \bullet 20}{100^2} = 0.04$$

$$p_e = 0.64 + 0.04 = 0.68$$

$$\kappa = \frac{0.92 - 0.68}{1 - 0.68} = 0.75$$

A Cohen's Kappa score of 0.75 is exceptional for a simple model and signifies a high performance in our methods. We consider this performance to be satisfactory and choose not to tune *k* further as a result. Words and phrases are clustered intuitively into ideas, and this success is then embedded in our generative model, showing great promise for idea-based tokenization methods like this.

Conclusion

Through a method of clustering word embeddings into clusters representative of ideas, a generative model was created, with high performance for its small size. The clusters themselves were evaluated with Cohen's Kappa and found to represent ideas very well. With simple methods – k-means clustering, creating word embeddings with a pre-trained Word2Vec model, and a next-token generator with an n-gram – performant results were achieved towards the goal of idea generation. This project has high potential to scale in the future, with many of its weak points due to the lack of data as a result of the limited computational resources available for the project. However, through clustering high-dimensional word embeddings to represent ideas and complex concepts for generative language models serves as a novel approach to idea generation which can prove to be very interesting and impactful once scaled.

On a personal level, I have a deep interest in AI-driven software and am pursuing a career in the intersection of the two. I can imagine software with language models "thinking" asynchronously to contribute to the inspiration of their human users – not too far off from a scaled version of this project. I had a lot of fun building this model and learned many new skills from implementing Cohen's Kappa, building a next-token generator for the first time at this scale, and pushing the boundaries of my laptop's RAM running the code. I also find concepts of knowledge representation and manipulating models' latent representation space very interesting, so being able to build out an idea I had to better represent knowledge in generative models was very exciting. I had the insight through this project that some of the most intelligent speakers are very precise with their language which helps to make their points very clear. If I were to expand this project further I would focus on exactly that – scaling to the point of making each idea cluster extremely narrow and precise with suitable data to support generation. All in all, a lot of highly applicable lessons were learned that have given me some crucial insight for what's important to think about when building language models in the future.