Summarize Podcast Transcripts and Long Texts Better with NLP and AI

Why the existing summarization approach is flawed, and a walkthrough of how to do better

Isaac Tham · Follow

Published in Towards Data Science

11 min read · May 3

Image from Unsplash.

LLMs like GPT-4 have taken the world by storm, and one of the tasks that generative text models are particularly good at is summarization of long texts such as books or podcast transcripts. However, the conventional method of getting LLMs to summarize long texts is actually fundamentally flawed. In this post, I will tell you about the problems with existing summarization methods, and present a better summarization method that actually takes into account the structure of the text! Even better, this method will also give us the text's main topics — killing two birds with one stone!

I will walk you through how you can easily implement this in Python, with just several tweaks of the existing method. This is the method that we use at Podsmart, our newly-launched AI-powered podcast summarizer app that helps busy intellectuals save hours of listening.

Problems with existing solutions

The canonical method to summarize long texts is by **recursive summarization**, in which the long text is split equally into shorter chunks which can fit inside the LLM's context window. Each chunk is summarized, and the summaries are concatenated together to and then passed through GPT-3 to be further summarized. This process is repeated until one obtains a final summary of desired length.

However, the major downside is that existing implementations e.g. LangChain's summarize chain using map_reduce, split the text into chunks with no regard for the logical and structural flow of the text.

For example, if the article is 1000 words long, a chunk size of 200 would mean that we would get 5 chunks. What if the author has several main points, the first of which takes up the first 250 words? The last 50 words would be placed into the second chunk with text from the author's next point, and passing this chunk through GPT-3's summarizer would lead to potentially important information from the first point being omitted. Also, some key points may be longer than others, and there is no way of knowing this a priori.

Another method is the 'refine' method, which passes every chunk of text, along with a summary from previous chunks, through the LLM, which progressively refines the summary as it sees more of the text (see the prompt here). However, the sequential nature of the process means that it cannot be parallelized and takes linear time, far longer than a recursive method which takes logarithmic time. Additionally, intuition suggests that the meaning from the initial parts will be overrepresented in the final summary. For podcast transcripts where the first minutes are advertisements completely irrelevant to the rest of the podcast, this is a stumbling block. Hence, this method is hence not widely used.

Even if more advanced language models come out with longer context windows, it will still be woefully inadequate for many summarization use cases (entire books), and some chunking and recursive summarization is inevitably necessary.

In essence, if the summarization process doesn't recognize the text's hierarchy of meaning and isn't compatible with it, it's not likely that the resulting summary will be good enough to accurately convey the author's intended meaning.

A Better Way Forward

A better solution is to tackle the summarization and topic modelling process together in the same algorithm. Here, we split the summary outputs from one step of the recursive summarization into chunks to be fed into the next step. We can achieve this through by clustering chunks semantically into topics and passing topics into the next iteration of the summarization. Let's walk you through how we can implement this in Python!

Requirements

Python packages:

- scipy — for cosine distance metric
- networkx — for the Louvain community detection algorithm
- langchain — package with utility functions allowing you to call LLMs like OpenAI's GPT-3

Data and Preprocessing

The GitHub repository with the Jupyter notebook and data can be found here: https://github.com/thamsuppp/llm_summary_medium

The text we are summarizing today is the 2023 State of the Union speech by US President Joe Biden. The text file is in the GitHub repository, and here is the original source. The speech, as are all US government publications, are in public domain. Do note that it is important to make sure that that you are allowed to use the source text — Towards Data Science has published some helpful tips about checking for dataset copyrights and licenses.

We split the raw text it into sentences, restricting sentences to have a minimum length of 20 words and maximum length of 80.

Creating Chunks

Instead of creating chunks large enough to fit into a context window, I propose that the chunk size should be the number of sentences it generally takes to express a discrete idea. This is because we will later embed this chunk of text, essentially distilling its semantic meaning into a vector. I currently use 5 sentences (but you can experiment with other numbers). I tend to have a 1-sentence overlap between chunks, just to ensure continuity so that each chunk has some contextual information about the previous chunk. For the given text file, there are 65 chunks, with an average chunk length is 148 words, with a range from 46–197 words.

Getting Titles and Summaries for Each Chunk

Now, this is where I start deviating from LangChain's summarize chain.

Getting two for the price of 1 LLM call: title + summary

I wanted to get both an informative title as well as a summary of each chunk (the importance of the title will become clearer later). So I created a custom prompt, adapting Langchain's default summarize chain prompt. As you can see in `map_prompt_template - text` is a parameter that will be inserted into the prompt - this will be the original text of each chunk. I create a LLM, which is currently GPT-3, and create an LLMChain, which combines an LLM with the prompt template. Then, `map_llm_chain.apply()` calls GPT-3 with the prompt template with the text inputs inserted in, returning titles and summaries for each chunk, which I parse into a dictionary output. Note that all chunks can be processed in parallel as they are independent of each other, hence leading to immense speed benefits.

You can use ChatGPT for 10x cheaper price and similar performance, however when I tried it, only the GPT-3 LLM runs the query in parallel, whereas using ChatGPT runs it one-by-one, which was painfully slow as I normally pass in ~100 chunks at the same time. Running ChatGPT in parallel requires an async implementation.

```
def summarize_stage_1(chunks_text):

  # Prompt to get title and summary for each chunk
  map_prompt_template = """Firstly, give the following text an informative title.
  {text}
```

```
    Return your answer in the following format:
    Title | Summary...
    e.g.
    Why Artificial Intelligence is Good | AI can make humans more productive by aut
    TITLE AND CONCISE SUMMARY:"""
    map_prompt = PromptTemplate(template=map_prompt_template, input_variables=["tex
    # Define the LLMs
    map_llm = OpenAI(temperature=0, model_name = 'text-davinci-003')
    map_llm_chain = LLMChain(llm = map_llm, prompt = map_prompt)
    map_llm_chain_input = [{'text': t} for t in chunks_text]
    # Run the input through the LLM chain (works in parallel)
    map_llm_chain_results = map_llm_chain.apply(map_llm_chain_input)
    stage_1_outputs = parse_title_summary_results([e['text'] for e in map_llm_chain
    return {
      'stage_1_outputs': stage_1_outputs
    }
```

Embedding Chunks and Clustering into Topics

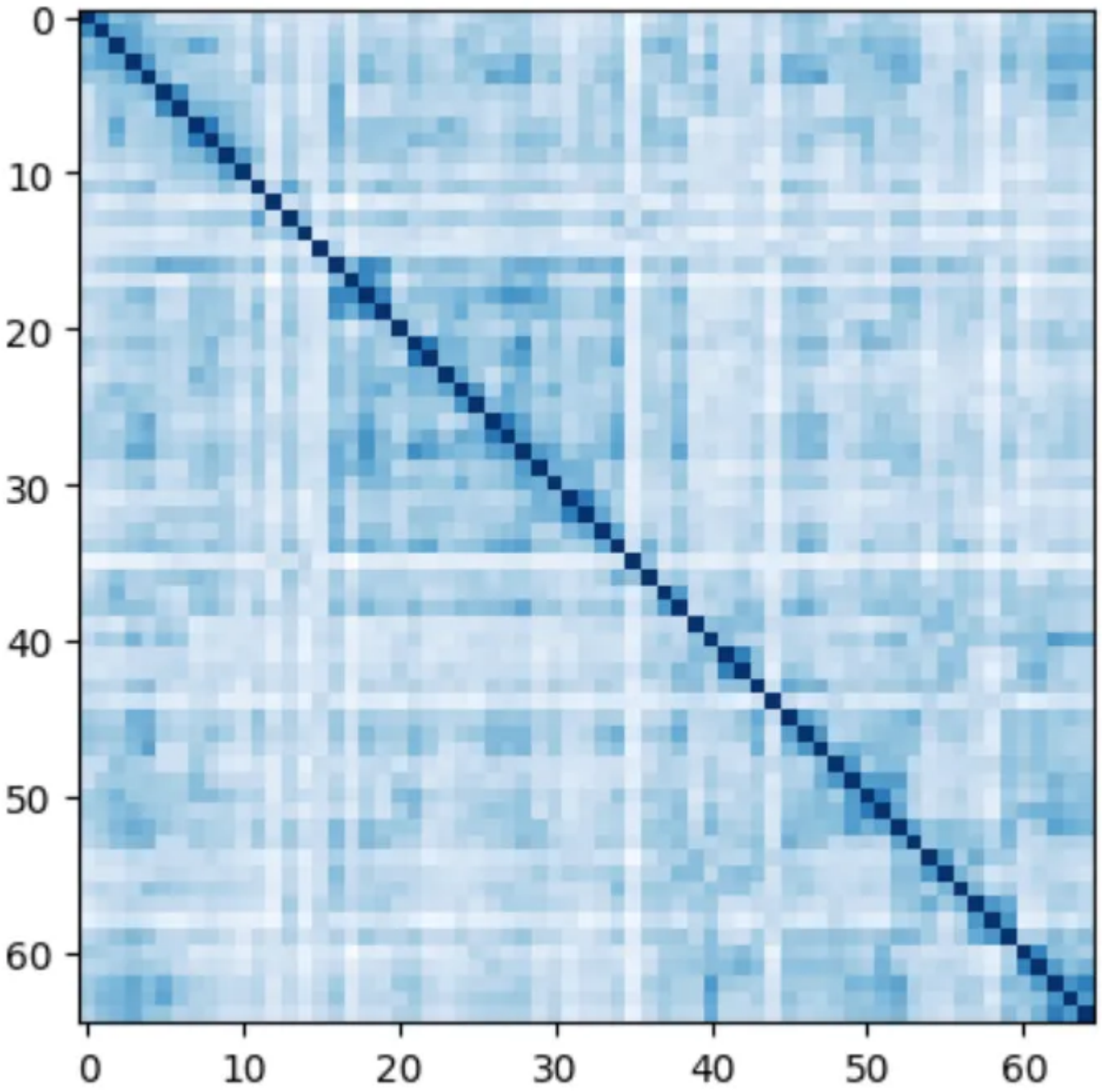
After obtaining the summaries for each chunk, I will embed them using OpenAI's embeddings into 1536-dimension vectors. The conventional recursive summarization method does not require embedding as they split texts arbitrarily by even length. For us, we aim to improve on that by grouping semantically-similar chunks together into topics.

Grouping texts into topics is a well-studied problem in NLP, with many traditional methods such as Latent Dirichlet Allocation which predates the age of deep learning. I remember using LDA in 2017 to cluster newspaper articles for my college's newspaper — it was very slow to estimate, and only used word frequency which does not capture semantic meaning.

Now, we can leverage OpenAI's embeddings-as-a-service API to obtain embeddings that capture the semantic meaning of sentences in one second. There are many other possible embedding models that can be used here e.g. HuggingFace's `sentence-transformers`, which reportedly has better performance than OpenAI's embeddings, but that involves downloading the model and running it on your own server.

After obtaining embedding vectors from the chunks, we group similar vectors together.

I create a chunk similarity matrix, where the `(i,j)th` entry denotes the cosine similarity between the embedding vectors of the ith and jth chunk, i.e. the semantic similarity between the chunks.



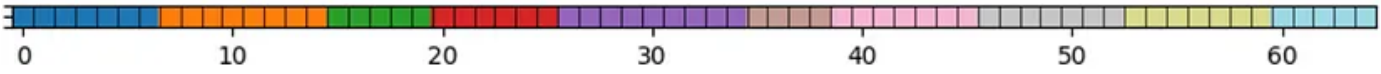
The chunk similarity matrix for the State of the Union speech. You can see certain groups of chunks are similar to collectively similar to each other — this is what the topic detection algorithm later on will uncover. Image by author.

We can view this as a similarity graph between nodes which are chunks, with edge weight being the similarity between two chunks. We use the Louvain community

detection algorithm to detect topics from the chunks. This is because communities are defined in graph analysis as having dense intra-community connections, and sparse inter-community connections, which is what we want: chunks within a topic to be very semantically-similar to each other, while each chunk being less semantically-similar to chunks in other detected topics.

The Louvain community detection algorithm has a hyperparameter called resolution — small resolutions lead to smaller clusters. Additionally, I add a hyperparameter `proximity_bonus` - which bumps up the similarity score of chunks if their position in the original text is closer to each other. You can interpret this as treating the temporal structure of the text as a prior (i.e. chunks closer to each other are more likely to be semantically similar). I put this in to discouraging the detected topics from having chunks from all over the text which is less plausible. The function also tries to minimize the variance in cluster sizes, preventing situations when one cluster has 1 chunk while another has 13 chunks.

For the State of the Union speech, the output are 10 clusters, which are nicely continuous.



Detected topic clusters for the State of the Union speech. Image by author.

Detected topic clusters for a Bloomberg Surveillance podcast transcript. The purple and orange topics picked up advertisements. Image by author.

The second image is the topic clustering for another podcast episode. As you can see, the beginning and end is detected to the same topic, which is common for podcasts with ads at the start and end of the episode. Some topics, like the purple one, are also discontinuous — it's nice that our method allows for this, as a text can cycle back to an earlier-mentioned topic, and this another possibility that the conventional text-splitting fails to account for.

Topic Titles and Summaries

Now, we have topics that are semantically coherent that we can pass into the next step of the recursive summarization. For this example, this will be the last step, but for much longer texts like books, you can imagine repeating the process several times until there are ~10 topics left whose topic summaries can fit into the context window.

The next step involves three different parts.

Topic Titles: For each topic, we have generated a list of titles for the chunks in that topic. We pass all topics' list of titles into GPT-3 and ask it to aggregate the titles to arrive at one title for each topic. We do this concurrently for all topics to prevent the topics' titles from being too similar with one another. Previously, when I generated topic titles individually, GPT-3 does not have context of the other topic titles, hence there were cases where 4 out of 7 titles were 'Federal Reserve's Monetary Policy'. This is why we wanted to generate chunk titles — trying to fit all chunk summaries into the context window here may not be possible for very long texts.

As you can see below, the titles look good! Descriptive, yet unique from each other.

```
1. Celebrating American Progress and Resilience
2. US Economy Strengthening and Inflation Reduction
3. Inflation Reduction Act: Lowering Health Care Costs
4. Confronting an Existential Threat: Making Big Corporations Pay
5. Junk Fee Prevention Act: Stopping Unfair Charges
6. COVID-19 Resilience and Vigilance
7. Fighting Fraud and Public Safety
8. United States' Support for Ukraine and Global Peace
9. Progress Made in Healthcare and Gun Safety
10. United States of America: A Bright Future
```

Topic Summary: Nothing new here, this involves combining the chunk summaries of each topic together and asking GPT-3 to summarize them into a topic summary.

Final Summary: To arrive at the overall summary of the text, we once again concatenate the topic summaries together and prompt GPT-3 to summarize them.

President Biden has congratulated the members of the 118th Congress, including the new Speaker of the House, Kevin McCarthy, and the new House Minority Leader, Hakeem Jeffries. He has celebrated the progress and resilience of the US, which has created 12 million new jobs in two years. Democrats and Republicans have come together to pass significant laws, with the goal of restoring the soul of the nation and rebuilding the middle class. The Inflation Reduction Act is saving millions of people $800 a year on their premiums, and the Bipartisan Infrastructure Law is the largest investment in infrastructure since President Eisenhower's Interstate Highway System. President Biden is also investing in clean energy, electric vehicle charging stations, and conservation efforts. The Junk Fee Prevention Act is proposed to protect workers from unfair fees, and the President is taking steps to reduce child poverty and increase economic growth. He is also advocating for the Act, which would give Medicare the power to negotiate drug prices. Additionally, he is taking a stand against Putin's war against Ukraine and is committed to competing with China. Finally, he is calling for a ban on assault weapons and for bipartisan action on immigration, and for us to embrace light over darkness, hope over fear, and unity over division.

The final summary of the State of the Union address. Image by author.

To summarize

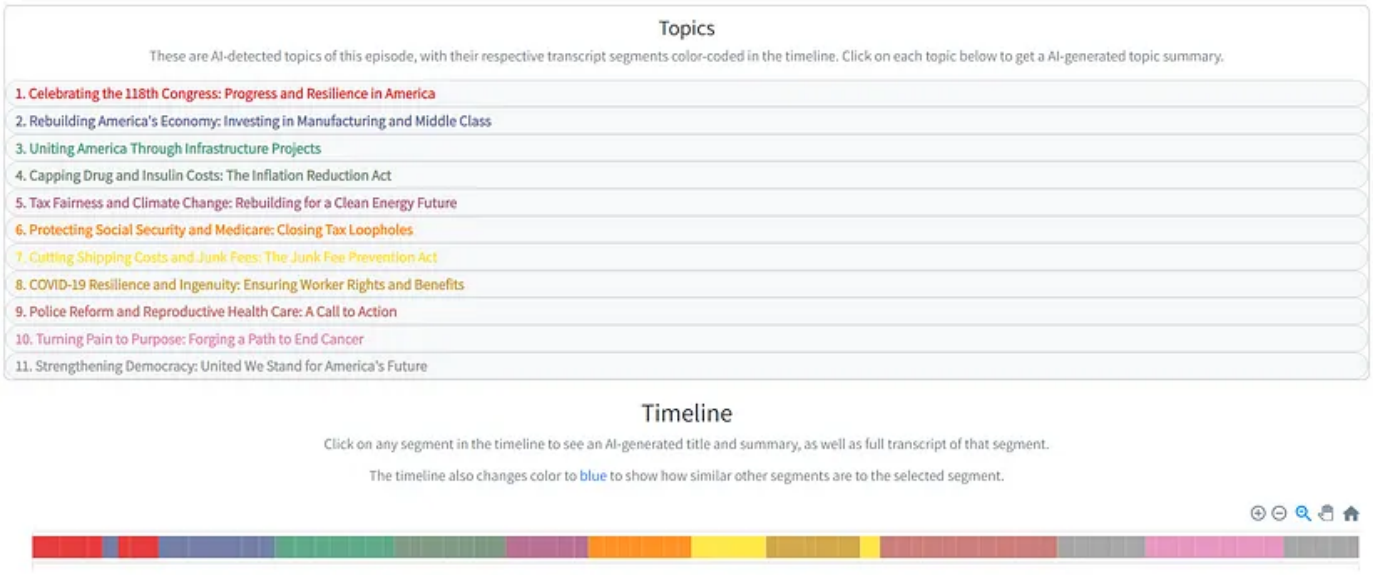
What are the benefits of our method?

The text is split hierarchically into topics, chunks, and sentences. As we progress down the hierarchy, we get progressively detailed and specific summaries, from the final summary, to each topic's summary, to each chunk's summary.

As I mentioned above, the summary hence accurately captures the semantic structure of the text — where there is an overarching theme which is split into several main topics, each of which comprises several key ideas (chunks), ensuring that the essential information is retained through the various layers of summarization.

This also offers greater flexibility than merely an overall summary. Different people are more interested in different parts of the text and would hence choose the appropriate level of detail they want for each part of the text.

Of course, this requires pairing the generated summaries with an intuitive and coherent interface that visualizes this hierarchical nature of the text. An example of such a visualization is on Podsmart— click here for an interactive summary of the speech.



A visualization of the extracted topics and the timeline of the transcript. Image by author.

Note that this does not drastically increase the LLM costs — we are still passing just as much input as the conventional method into the LLM, yet we get a much richer summarization.

TLDR — here are the secret sauces to produce superior summaries of your texts

1. Semantically-coherent topics — by doing semantic embeddings on small chunks of the text and splitting the text by semantic similarity
2. Obtaining titles and summaries from chunks — which required customizing the prompt instead of using the default LangChain summarize chain
3. Calibrating the Louvain community-detection algorithm — hyperparameters like resolution and proximity bonus ensure the generated topic clusters are plausible
4. Distinct topic titles — concurrently generating all topic titles which required chunk titles

Once again, you can check out the entire source code on the GitHub repo. If you have any questions, feel free to reach out to me on Twitter!

Artificial Intelligence | NLP | Podcast | Machine Learning | Hands On Tutorials

Follow

Written by Isaac Tham

205 Followers · Writer for Towards Data Science

economics enthusiast, data science devotee, f1 fanatic, son of God

More from Isaac Tham and Towards Data Science

Isaac Tham in Towards Data Science

Generating Music using Deep Learning

Introducing a new VAE-based architecture to generate novel musical samples

15 min read · Aug 25, 2021

493 3

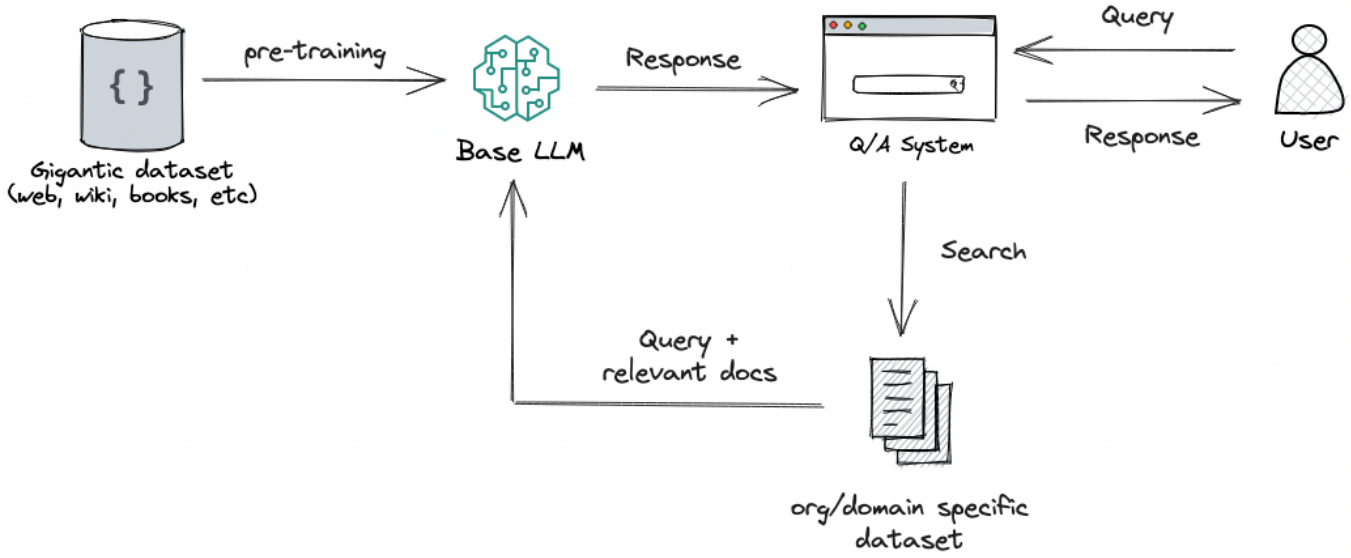
Giuseppe Scalamogna in Towards Data Science

New ChatGPT Prompt Engineering Technique: Program Simulation

A potentially novel technique for turning a ChatGPT prompt into a mini-app.

9 min read · Sep 4

1.5K 14



Heiko Hotz in Towards Data Science

RAG vs Finetuning — Which Is the Best Tool to Boost Your LLM Application?

The definitive guide for choosing the right method for your use case

· 19 min read · Aug 25

2.2K 16

Isaac Tham

Podsmart: Summarize Podcasts with AI

Podsmart summarizes podcasts with AI to help busy intellectuals learn more efficiently. We transcribe podcasts using latest AI models.

6 min read · Apr 22

24 1

See all from Isaac Tham

See all from Towards Data Science

Recommended from Medium

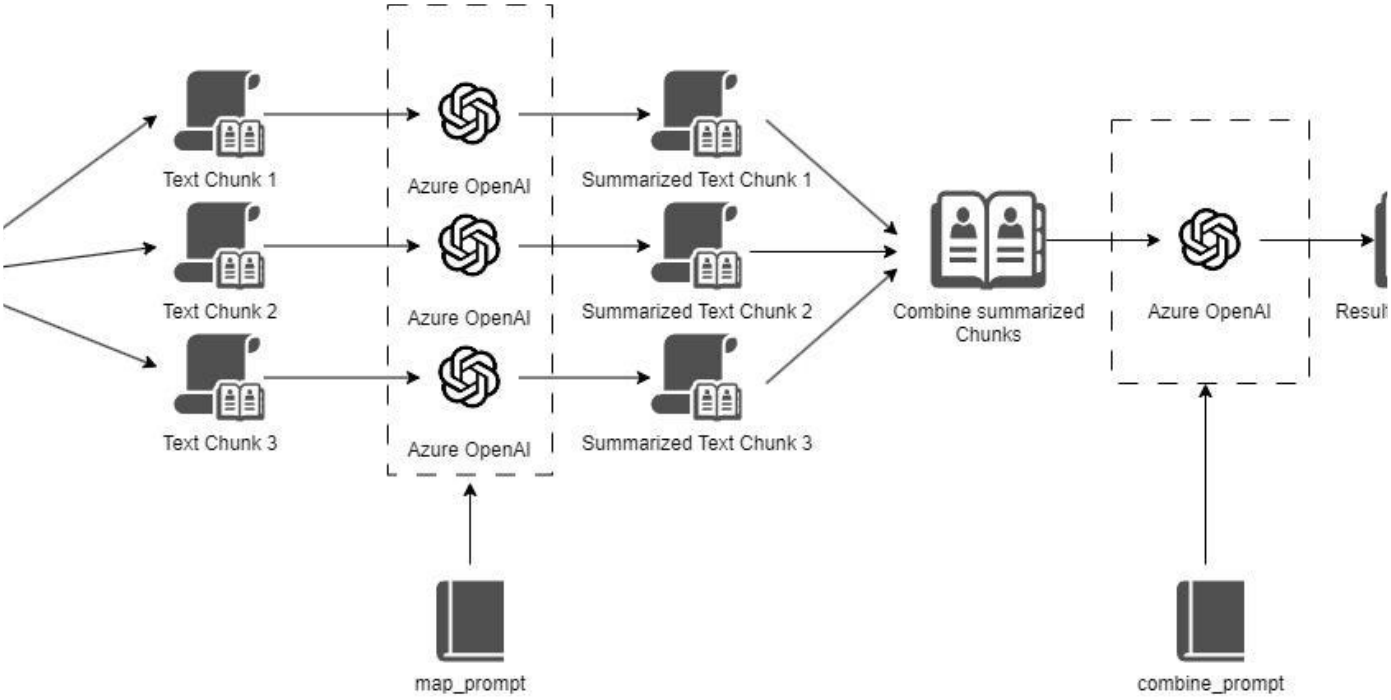
Thu Ya Kyaw in Google Cloud - Community

How to Use LLMs to Generate Concise Summaries

Large language models (LLMs) are a type of artificial intelligence (AI) that can be used to generate text. They are trained on massive...

4 min read · May 29

27 2



Bluetick Consultants

Advanced Techniques in Text Summarization: Leveraging Generative AI and Prompt Engineering

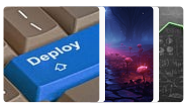
Generative AI and Language Models like OpenAI's GPT-3.5 have emerged as powerful tools for text summarization. In this blog, we will explore

9 min read · Jun 3

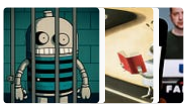
21 1

Lists

Natural Language Processing
657 stories · 265 saves

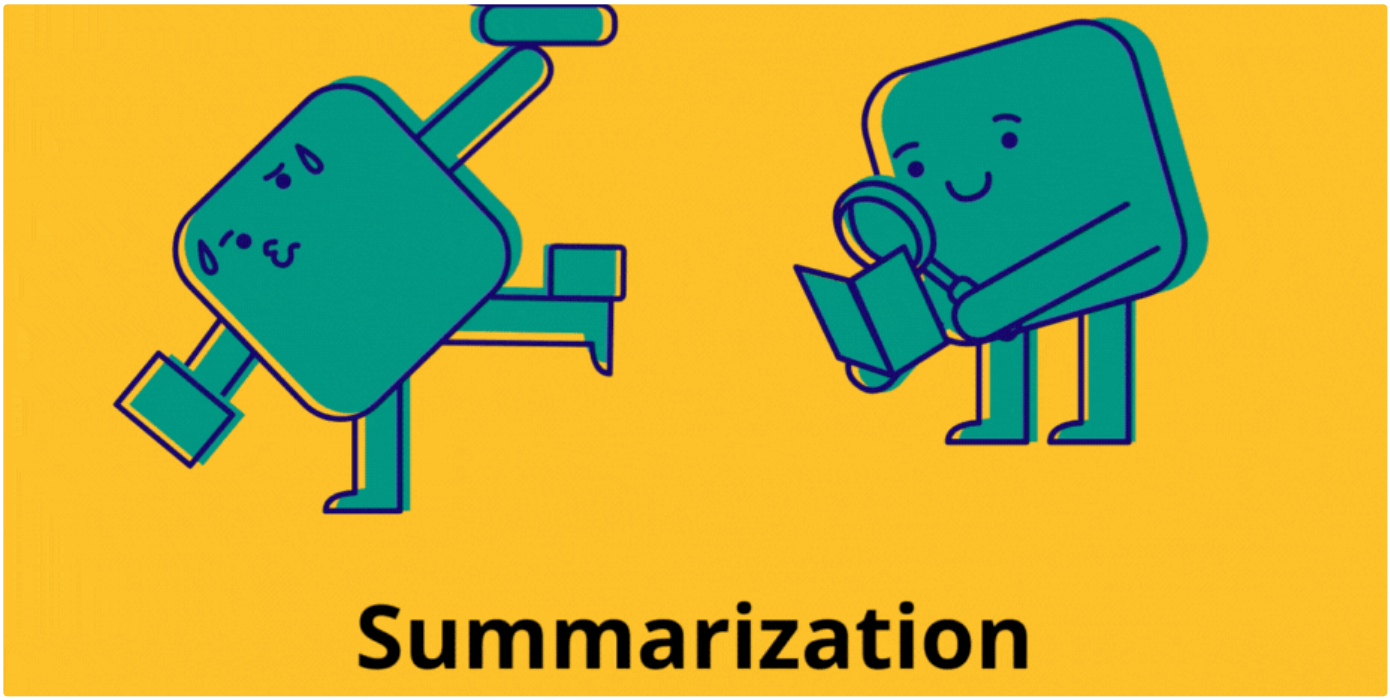


Predictive Modeling w/ Python
20 stories · 431 saves



AI Regulation
6 stories · 137 saves

ChatGPT prompts
24 stories · 439 saves



Andrea Valenzuela in Towards Data Science

Mastering ChatGPT: Effective Summarization with LLMs

How to Prompt ChatGPT to get High-Quality Summaries

· 10 min read · May 22

Paolo Rechia in Better Programming

Building a Question-Answer Bot With Langchain, Vicuna, and Sentence Transformers

A Q/A bot with open source

10 min read · May 1

 500  4 

Abel K Widodo

Exploring the Art of Generative AI in Python

Artificial Intelligence (AI) has transformed various industries, and one fascinating area of AI is generative AI. Generative AI involves...

6 min read · Jul 15

1

Aseem Srivastava

Dialogue Summarization

Here is a developing blog on the latest papers on dialogue summarization that I found relevant during my research on Dialogue Systems as a...

1 min read · Apr 2