Hierarchical [Multi-level] Summarization with Semantic Retrieval

Version 1/20/2025

Type: Documentation/Proposal

Author: Ryan Nader

Berkeley Artificial Intelligence Research (BAIR) Lab

University of California, Berkeley

Last Modified: Jan 20, 2025

V3
To summarize my status for hierarchical [multi-level] summarization for semantic retrieval at a HIGH LEVEL (without all details), I broke it up into parts.
...

- Building the tree which is responsible for organizing the information
- Tagging the tree which is responsible for allowing the search system to work
- Building and training the semantic retrieval system (i.e. search system)

...
NOTE: Also please note it is all being designed currently so some things might be removed/added/changed and feel free to critique or ask questions during literature review.

- Abstract and Project Overview
    - Key Idea (IMPORTANT)
        - this is the organizational system
            - hierarchical [multi-level] summarization → breaking down and structuring a large text into summaries at varying levels of detail.
        - this is the search system
            - semantic retrieval → method of information retrieval that focuses on understanding the meaning (semantics) of queries in documents, rather than relying solely on keyword matching
        - $\implies$ this implies that this project is an organization system with an advanced command find for large sets of documents
        - This system will create an organized layout of document information, allow input questions, and return the estimated location of the exact document answer and provide the user with this allowing the user to have high likelihood locations to start searching to find information.
        - If the answer is not present it will say not present, the idea is you **must** find the answer from the document as well as the location rather than from the language models training dataset, which may generate incorrect results.
        - This is key because if we assume the document has the correct answer then this should find it and be no more or less correct than the document itself.
    - Definitions
        - Sentence Embeddings
            - They are vector representations of sentences in a high-dimensional space, capturing semantic meaning numerically.
            - Foundational for a lot of the processes in this project.
        - Redundancy
            - This ensures that summaries at each hierarchical level are concise, non-repetitive, and avoid including multiple sentences conveying the same information that may appear often.
            - Redundancy can be minimized by maximizing pairwise cosine similarity between sentence embeddings and removing sentences with high similarity scores.
            - Functional Form: TODO
        - Salience
            - This helps identify which sentences within a cluster are most valuable for inclusion in the summary.
            - Salience can be modeled by scoring sentences based on their centrality in the embedding space (e.g., cosine similarity to the cluster centroid).
            - Functional Form: TODO
        - Coherence
            - This ensures that the summary flows logically, with sentences connected in a way that makes sense to the reader and maintains narrative structure.
            - Coherence can be measured by evaluating sequential similarity between embeddings of adjacent sentences.
            - Functional Form: TODO
        - Clustering
            - This groups sentences with similar semantic meanings together, allowing each cluster to represent a single idea or topic.
            - Clustering can be performed using methods like k-means, agglomerative clustering, or spectral clustering on sentence embeddings.
            - The number of clusters can be determined based on the desired granularity of the summary.
            - Functional Form: TODO
        - Vertical Checking
            - When using this as adjectives (for instance vertical coherence) I mean checking coherence on sequential sentences on the same level of the tree.
        - Horizontal Checking
            - When using this as adjectives (for instance horizontal coherence) I mean checking coherence on a parent node a summary with its associated child sentences one layer apart.
- Problems this addresses
    - Language models when trained are often able to answer questions by giving the response with the highest probability of sequential characters yet this has some issues.
    - It is difficult to determine where that answer came from.
    - If the user wants to add their data most language models struggle to handle large amounts of text as they have limited context size.
    - Furthermore, once the input becomes too large the model is less able to provide all details in the response.
    - Once an answer is given users have no freedom to explore the document itself and look at estimated areas of where the answer could be within the document if they are not satisfied.
    - Even if the model is given a document and can take it as input there is no effective way to control hallucination without asking the question a lot and manually checking with the document itself.
    - Even if a language model can be trained on that document, it will most likely be (1) more expensive to build instead of taking advantage of prebuilt language models and (2) it will need more data to understand language and is again prone to the black box of hallucination in the output.
    - Organization and learning theory
        - we also considering exploring the best way to organize information; for example, following link and link (along with others) building on past research looking at how we fundamentally organize information in our brain and then trying to use this system to organize our information so to provide users with minimum text to brain translation overhead to lower information latency (still early brainstorming this idea though and not essential)
    - Requirements of the model design
        - Data structure system
            - we need to have a 1 to 1 mapping of document text to answers and it must also be the exact text if that is what the user requests, this should work 100% of the time
            - we want a highly accurate multi-level summarization of every document that is added, ideally being able to handle common document types (this most likely will need to be continuous English sentences to work well, like books and mostly continuous text-based documents but generalizing this further would be ideal)
        - Interface
            - we want to allow users to explore groups of documents hierarchically, where the hierarchy can be metric-based as well as get a ChatGPT-like answer as well if requested
            - we want the exploration to be responsive
        - Answering system
            - we want a semantic retrieval system (some kind of traverse algorithm) that can take in questions, give locations of potential answers, and put the user interface in this location so the user can quickly see all the answers and self-navigate from there
            - we also understand that language models are really good at taking in small amounts of text and answering from there so if we can find a subset of the sentences where the answer is and then feed that into a language model that would also be a function so users can have a traditional language model response if they do not want to navigate the tree with mappings to tree nodes
- Estimated Progress for Presentation Date (January 27, 2025)
    - I should be able to provide my design and make some slides by then but am unsure about if I can get all the implementation and results by then.

- Currently, I have just been running on my local system and so far implementation-wise, LM requests are the main time bottleneck as I am running meta-llama/Llama-2-7b-chat-hf via VLLM with –cpu-offload-gb.
- This runs the model at approximately 120 seconds to 500 seconds per request depending on the request.
- I have a trivial/demo implementation of just building the text part of the tree with this model on Chapter 1: Loomings from Moby Dick, with a demo website (as a rough sanity check, website) where you can expand the hierarchical system that is created.
- Another concern for me is how long it will take to set up the server once I gain access.
- Also, note right now my main goal is to test my ideas (for the whole system) on 1 to 3 books at most and make sure it all is possible. I am currently working on the Multi-Level Tree section.

- Multi-Level Tree
  - The key idea here is that we want to build the tree in reverse order to maximize the coherence metric.
  - We want to maximize the coherence for both vertical and horizontal moves i.e. each level should be coherent as you should be able to read all the sentences in order.
  - My understanding is that this idea is similar to the paper, link which defines parent-to-child coherence and intra-cluster coherence, the notable difference; however, is we will test not just top-down but also bottom up as the paper states "we perform hierarchical clustering top-down" and what we want to test is if there is a way to get better coherence with this approach.
  - This component of the design is critical to the system's ability to match answers with the document information. If this is wrong we might be able to find the correct answer if other parts work; however, those answers will not then match the document information. So we need this to make sure all facts match the document.
  - To scientifically show this we will probably need to implement both and analyze the differences.
  - Along with this we also want to handle the other metrics they use like: salience, handling redundancy, and clustering which this approach may or may not improve.
  - Experiment 1
    - It is a tree with 4 levels (level 0, 1, 2, 3) and has:
      - level0=100 lines/sentences → which is Moby Dick's direct text, there are 100 sentences in this chapter
      - level1=100 lines/sentences → which is Moby Dick's normalized text
        - this normalized text layer could be removed if we just want the document text but it is here for two reasons
          - It converts text to analytic, simple English which makes it easier to read
          - It normalizes the text and due to different documents being written in different time periods, ideally having them on a common tone should remove potential issues we might have when finding content
            - for instance, we might perform better on books 1900 to 2024, while worse on 1800 to 1899, I am thinking of this as normalizing a dataset before training a classifier (although not all need the dataset normalized that is the inspiration)
          - I also like this as it makes my life easier as it is a lot easier to read documents in modern English
      - level2=20 lines/sentences → I compress 5 sentences into one, and summarize it
      - level3=4 lines/sentences → Again I compress 5 sentences into one and summarize it
      - *I write it as lines/sentences because my parser is still being updated so sometimes it takes more than one sentence depending on the format
      - During this experiment, I read through each level to see how much sense it made just from a useability standpoint and found that level 1 seemed pretty good in my mind around 95/100 lines read well.
      - Level 2 and above have some issues but as this is a trivial approach I feel that it has potential to be improved.
      - I am currently working on getting numerical metrics
    - Future idea for next experiment (node-level sentence optimization)
      - The plan to improve this is to modulate the $c = \text{CompressionFactor}$ as well as the $p = Promt$ and the $l = \text{LanguageModel}$ itself this will give us multiple sentences to choose from.
      - We can call this set of sentences $S = \{\}$.
      - Each of these sentences will have a score that is a linear combination based on variables like coherence (vertical and horizontal), salience, and redundancy. We will have an objective function that operates both vertically (on the same level) and horizontally (across levels) and the goal of this objective will be to pick the optimal series of sentences overall.
      - Think about this as increasing our probability of having the theoretical best sentences by having more possibilities to choose from.
    - Issues
      - Currently, my LM requests to build this are in the sum of all the nodes in the tree, so for this example, it would be $100 + 20 + 4 = 124$.
      - If we approximate the runtime with just the LM requests which are [120, 500] seconds it is upper bounded by: $124 \times 500 = 62000$ seconds which is around 17 hours (this is with a slower GPU and overestimates as most requests are on the lower end).
      - Now this builds Chapter 1 of Moby Dick with the reduction/merging of 5 sentences summarized per level, this is a free variable and is one factor we will need to modulate for the paper alongside the prompts and models used.
      - To put this in perspective Moby Dick has around 8617 lines/sentences that we would build the tree from (to simplify the math let us say 8615), this would yield $\approx 8615 + 1723 + 345 + 69 + 15 + 3 = 10,770$ nodes and when multiplying this by 5 we have $10,770 \times 500 = 5385000$ seconds which is around 62 days 20 hours on my system to process just the tree for all of Moby Dick.
      - This illustrates how critical it is to get the language model request runtime down; otherwise, we will have to decrease the granularity at which we process the documents, which will most likely lead to worse results. We will also need to test this.
      - The ideal goal is to get the entire runtime (both the building of the summarization and retrieval systems) to process each document in 1 to 3 hours per book, where each book is assumed the size of Moby Dick which is around 8617 lines/sentences and ideally this is on 1 GPU.
      - Furthermore, as each book can be processed in parallel for 1 hour, if we want to process 8000 document database, it will be 8000 hours ≈ 333.33 days, and with parallelization (16 GPUs) we can process this in around 21 days.
      - The worst case here is we potentially just use the OpenAI API but I have not looked into the cost of that or we also could drop the document amount to around 100 or something smaller.
      - Another issue was dealing with responses like "I apologize, but I cannot fulfill your request as the sentence you provided contains offensive language and cultural stereotypes. I am programmed to promote respectful and inclusive communication, and I cannot participate in normalizing or perpetuating harmful language or content."
      - Issues creating a parser that can consistently extract the sentences from the response.
      - Another issue is testing as we will probably want to try to do this while changing the free variables of the system, so I am right now thinking about doing this on just 5 books at most to tune the hyperparameters.
    - Positives
      - Each sentence can be run in parallel as they are all independent, so if we have the hardware this could be massive, again taking advantage of Amdahl's Law.
      - Also, all this parallelization might require Rust or another programming language (besides Python) where we will set up a distributed computing worker system to process all this.
      - I am hoping the runtime range on the server is significantly lower (no more than 60 seconds and 80% or more of the calls clustered around 5 to 10 seconds) and hoping to switch to a larger model, like meta-llama/Llama-3.1-8B or meta-llama/Llama-3.1-405B if possible.
      - Along with trying smaller models and specialized summarizer LM's, this will probably be even faster as I tried google-t5/t5-large and facebook/bart-large-cnn; however, the responses on these were very bad.

- Tagging Tree (think like netlist for PCB design)
  - So far I do not have this completely planned out, just ideas.
  - This next part is critical as we will use node tagging as a way to find the answer.
  - Note, that this is one idea that I came up with; however, we **need to read more papers** to identify if there have been past successes with this and if so maybe we prioritize those as this component is critical to the system's ability to find answers and assumes the answers it finds are correct.
  - PCB Analogy
    - As background, the idea behind this algorithm is copied from PCB design software.
    - One of the projects I worked on in the past was building a PCB.

- During this process, my understanding was in the past before fancy software electrical engineers would manually write all the connections on a schematic and painstakingly check each one, and when needing to find a net they would have to follow traces with their eyes (manually) to check it.
- Note nets are defined as electrical nodes or points that need to be connected in the circuit layout.
- This is how I am trying to think about groups of sentences, some are connected based on a metric, and right now documents still use the manual approach, to find connections we must read the text like an electrical engineer manually checking nets.
- However, one of the solutions electrical engineers came up with to solve this problem was to create a netlist in which the circuit designer essentially creates tags for all things that are connected (in this sense connected wires), and whenever you want to see things that are connect you can highlight all the connections.
- This organization of the netlist not only made the schematic design process quicker but also the PCB layout task quicker as being able to focus on the specific components you are connecting can allow a single person to thoroughly create a large schematic.
- Utilizing this tool is what allowed me to design my PCB which had around 2000 connections all within a semester.
- This idea or organization, in my opinion, is what makes the hardware space so interesting not only did it allow for a quicker layout but it also allowed for almost fully constraint-based optimized autorouting, where it could be almost entire autorouted, making a complex job in the past now a one person job.
- Furthermore, being able to have all the components sorted based on this netlist also made the process quicker.
- Right now to my understanding documents are still in that earlier stage, where you read one sentence at a time and we are trying to update the method to be like this design software where you can look at "nets" of useful connections.
- To finalize this idea I am calling nets equal to tags where matching tags are synonymous with building a netlist i.e. these are sentences that are "connected" by some metric.

- Search tag generation
  - these are the tags that will be used in the search/retrieval system
  - Now following this methodology the first thing that comes to mind is a tag where they are connected on sentence embeddings by using a hypersphere (n-dimension sphere) with a hyperparameter, radius r, and everything contained in this sphere is "connected."
  - Other ideas include the following.
    - Connected sentences by using a bag or words approach i.e. similar to command find (which I really believe has a useful place here) and linking sentences that have similar words or if possible creating a minimal set synonym word basis where we make all synonyms (like gleeful, joyful, cheerful, ...) mapped to some base form like happy and I think there could be a way to do this using a dictionary and thesaurus and just creating the space from that as by definition it contains the rules for English (especially from the perspective of if you think about English as a kind of programming language)
    - Also giving a language model 20 tags to choose from and asking it to pick the ones that are relevant for that group of sentences may also be another approach.
    - Using the norm and having a norm ball problem.
- Now in theory we should have search tags and the question becomes which nodes at which levels do we need tags for?
  - UNANSWERED, but right now just giving it to all of the nodes. TODO
- granularity search attribute/tag
  - This will be used to answer depth i.e. the semantic retrieval system creates a granularity value based on the question and will stop at that specific granularity. In theory, if the granularity is small enough we could just do a tree search on n-branching tree which would be $O(\log_n N)$ where there are N nodes if the tree is balanced.
    - Also, the tree could be resorted based on any of these tags.
  - Granularity will be between 0 and 1 where the raw text itself will be 1 i.e. the bottom of the tree, most granular.
  - Each node at the same level may have the same granularity the only exception being if we have a different $c = \text{CompressionFactor}$ at the same level i.e. one node n_1 at level l has summarized 3 sentences while another n_2 has summarized 10 sentences, then we would assume n_1 has higher granularity as it needs to condense less into one sentence.
  - Furthermore, it should be a function of how many base-level sentences it summarized as if the one below that summarized 100 sentences then it has less granularity than one that is a parent of a node that summarized only 20 in that stage.
  - This needs to be an attribute of the node in the class, how many leaf sentences it condenses, and that will determine the granularity for that node.
- So far this is what I have for this and if there are other ways that it is done then we can add that idea as tags that will be used and saved in a matrix T, the tags matrix and we will use this in the Building and Training section.
  - The tags are numerical or categorical metadata associated with nodes, representing their semantic topics or features.
  - In the matrix, each row corresponds to a specific node in the hierarchy.
  - For N total nodes in the hierarchy, there will be N rows.
  - Each column corresponds to a specific tag feature or dimension of the tag space.
  - For d tag features, there will be d columns.
  - We can use this to form the feature matrix.

- Building and Training the Semantic Retrieval System
  - For this section, I have two main ideas and this section needs to learn which path contains the answer based on the input question and tag matrix T.
  - How to get testing data with labels
    - I understand that this may not be the best method but the idea is as follows as I do not see another way to do it, if there are any ideas please suggest. TODO
    - To do this we will need training data and a simple way to get this is the ask a language model for each node come up with questions and multiple choice answers for this node and use that, if we ever have users and get it to work well enough then can have user feedback with reinforcement learning but we do not want to rely on this.
  - Method 1 (simple idea, definitely need to think more about how to do this as maybe there is a better way to learn the hierarchy like maybe feeding in the (96×1) 224 times and then going to the next sentence with the same label and making the input layer 96 nodes instead)
    - Train it with a classifier, (giving an example with a simple fully connected neural network) to learn the estimated path
    - Explicit Example Setup for training the retrieval system
      - For simplicity, we will assume no batching. Each pass is an one item group.
      - Also, assume ReLU activation.
      - Now we have 5 questions (number of questions: $Q = 5$), each represented as a 93-dimensional embedding vector ($\mathbf{q}_i \in \mathbb{R}^{93\times 1}$, so $\mathbf{q}_1$, ... $\mathbf{q}_5$).
      - Also, assume that the multi-level tree is like the Moby Dick example, $4 \to 20 \to 100 \to 100$ and this total number of nodes is 224 and they are just labeled $\{1, 2, 3, ...224\}$ in binary search way format to understand which way to navigate in log time.
      - And each node has 3 tags that are all numbers between 0 and 10 i.e. tags for each node: $t_n \in \mathbb{R}^3$, where each tag is a number between 0 and 10.
      - The corresponding labels are $\{23, 99, 44, 3, 224\}$ i.e. these are the answer nodes that should return that from our dataset i.e. labels: $y = \{23, 99, 44, 3, 224\}$, one for each question.
      - Input layer: 765 (93 from the question, then 3 from the tags $\times$ all the nodes which are 224 for the full flattened vector which gives 672, and thus the total input layer is 765), this simplifies what is happening and then we just have a (765×5) shaped matrix for all the sentences but we may lose the hierarchy relationships as I just giving best node, look at comment at Method 1 for more information on possible issues.
      - Hidden layers: use ReLU activation, could have $100 \to 50$, all these could be anything
      - Output layer: 224 (number of nodes), this is to match the output space with softmax.
      - Activation: ReLU for hidden layers, softmax for the output layer.
      - Overall Layers: $765 \to 100 \to 50 \to 224$
  - Construction
    - Question Embedding: Example for q_1 (shape 93×1):
      - $$\mathbf{q}_1 = \begin{bmatrix} 0.1 \\ 0.2 \\ \vdots \\ 0.5 \end{bmatrix}, \quad \mathbf{q}_1 \in \mathbb{R}^{93\times 1}$$
    - Tags for All Nodes: Tags for 3 nodes (out of 224):
      - $$\mathbf{T} = \begin{bmatrix} 5 & 3 & 8 \\ 2 & 10 & 1 \\ 4 & 7 & 6 \\ \vdots & \vdots & \vdots \\ 1 & 0 & 9 \end{bmatrix}, \quad \mathbf{T} \in \mathbb{R}^{224\times 3}$$

- Now we want to create a feature vector for each node n by combining the question embedding ($\mathbf{q}_1$) of size 93×1 and the tags (t_n) associated with that node, which is a vector of size 3×1. And $\mathbf{x}_{1,n} = \begin{bmatrix} \mathbf{q}_1 \\ \mathrm{t}_n \end{bmatrix}$ which is $\mathbf{x}_{1,n} \in \mathbb{R}^{96\times 1}$.
- Thus to create the feature matrix $\mathbf{X_1}$ for all nodes for question $\mathbf{q}_1$ we have $\mathbf{X}_1 = \begin{bmatrix} \mathbf{q}_1 \| \mathbf{t}_1 \\ \mathbf{q}_1 \| \mathbf{t}_2 \\ \vdots \\ \mathbf{q}_1 \| \mathbf{t}_{224} \end{bmatrix}$, $\mathbf{X}_1 \in \mathbb{R}^{224\times 96}$ i.e. this is 224 nodes with 96 features. And you would have one of these for each question q, meaning repeat for all questions $\mathbf{q}_2, \mathbf{q}_3, \ldots, \mathbf{q}_5$.
- For 5 questions, stack the flattened inputs: $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_5 \end{bmatrix}$, $\mathbf{X} \in \mathbb{R}^{5\times 765}$
- And the labels are $y^T = [23, 99, 44, 3, 224]$ which is shape (5×1).
- Key Ideas and Issues
    - The question embedding q_1 is global: it provides context for the model about the input query.
    - The tags t_n are local: they differentiate nodes from one another.
    - Thus when we concatenate and flatten them it gives the model both the global context (question) and the local node-specific information (tags) to decide which node is the best match, still not sure how to handle the hierarchy and learn that relationship or if this will be good enough.
    - Also, the input layer scales massively with node count, which may make this too large.
    - Also may need to look at other classifier algorithms and turn this into a Jupyter Notebook Kaggle problem where we have just training/test dataset and just pick one with the highest test accuracy on withheld data.
- In theory, the tags should provide additional context about nodes, helping the network disambiguate between nodes with similar question relevance.

- Method 2 (Abstract Still)
    - Use an existing LLM answering system or other existing ideas that do something similar.
    - Also maybe look into researching with ChatGPT o1 reasoning system maybe there is something there as well.
- END
    - Then once this is trained the idea is to expand the node where the answers are likely as well as make a language model request to answer the question. We also input that node data into the request along with the node number so the user can quickly find it and get 100% cited material.
    - Also, the beauty of the design is that we can build one of these for each document in parallel and once built the runtime becomes a tokenization, classifier (neural net) inference, tree traversal, and maybe a few language model requests which should be responsive.

- Reference Papers
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link
    - link