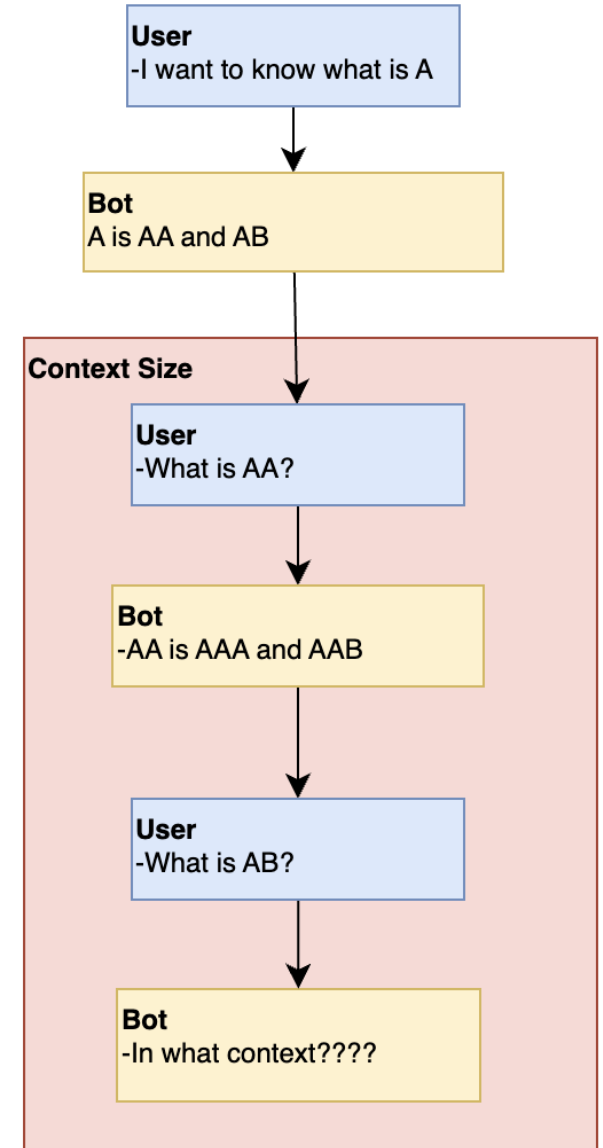


# Branch-chat

일관적이고 쾌적한 대화를 하기 위한 솔루션

# 문제 정의

- 대화를 하다가 세부 주제로 dive in, 혹은 새로운 주제로 넘어가기도 함.  
이후 다시 기존 주제로 돌아오려 하면,  
context 오염으로 인한 문맥 소실 발생
- 해결해야 하는 case
  - 입력했던 context를 재차 입력해야 함,
  - 과거 답변과 동일한 내용을 재차 출력으로 경험
- Attention과 큰 context size만으로는  
한 개인의 지속적인 전담 비서가 되기에는  
한계가 있음



# 활용 시나리오 정의

## • 기억을 요구하는 전문가 조수

- 보안 취약점 분석 중간보고
- → 새로운 고객 미팅 준비
- → 다시 이전 취약점 분석 맥락으로 복귀

## • 지속적 교육 및 개인화 튜터 시스템

- 챕터별로 branch 생성하며 질문
- 학생의 지식수준을 체계적으로 기억, 답변
- 학생이 학습 branch를 다시 이어가며 진도를 관리

## • AI 기반 고객지원 + 기술지원 시스템

- 고객의 목적 파악
- → 목적 달성을 위해 세부목표 달성
- → 다시 이전 본 목적으로 복귀

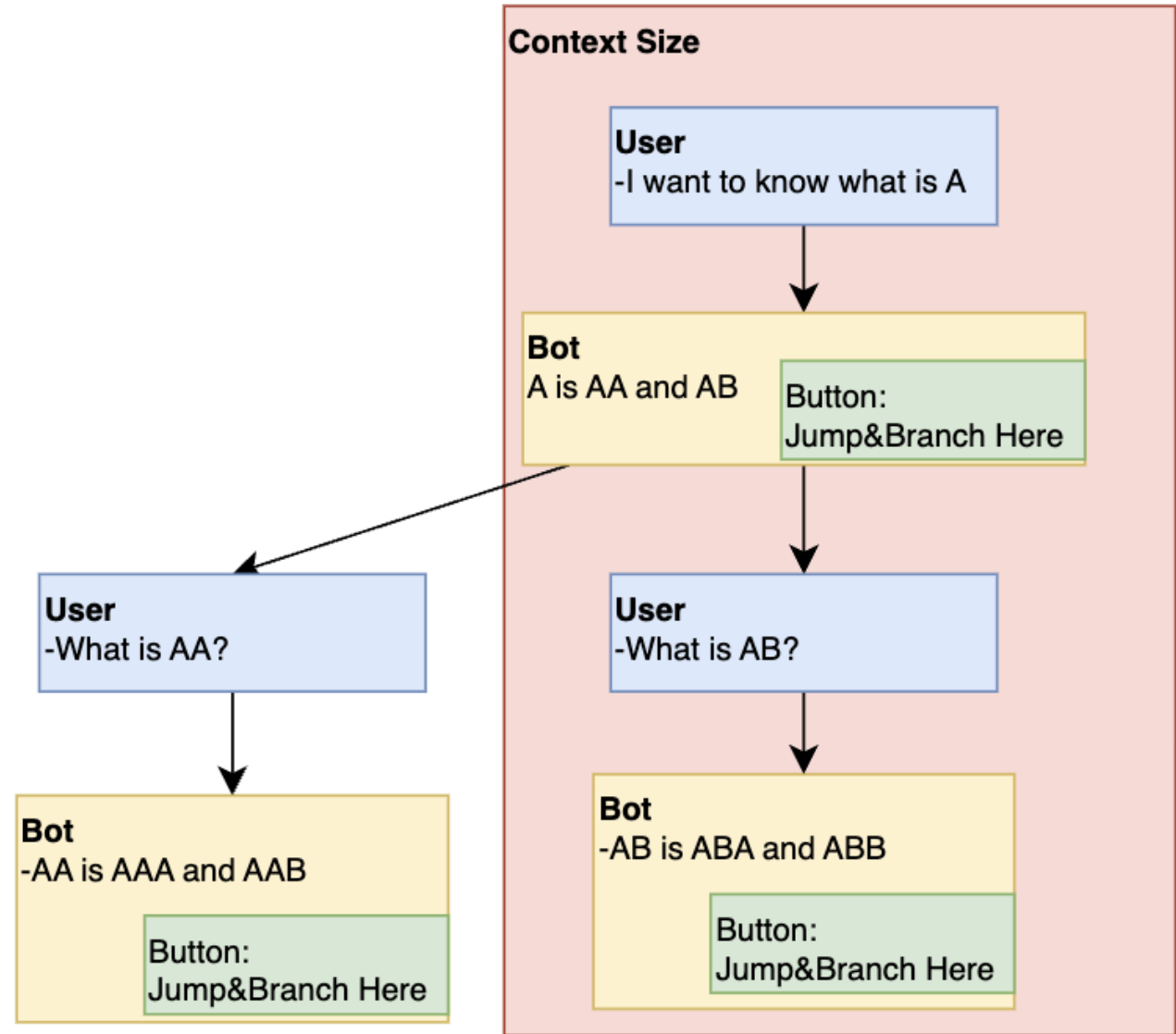
## • 개인 프로젝트 메모형 AI 어시스턴트

- 연구자가 논문 아이디어
- → 실험 설계 branch
- → 논문 작성 branch

# 구현 계획서

# 설계도

- 언제든지 과거 채팅 시점으로 jump
- 모든 branch 기억
  - 언제든지 기존 세부주제로 다시 checkout 가능
- 단순 context attention 의존을 극복
  - 사용자가 선택적으로 context를 관리할 수 있도록 함.
- 전체 대화 tree를 RAG에 삽입
  - 중요한 내용은 tree-wide로 이어지도록 유도.



# 관련 component

- 프로젝트 진행 제약사항
  - 시간의 부족
    - 모델들을 모두 다운 및 테스트하기엔 인터넷 속도 제약
  - 자료의 부족
    - 사내 정보를 테스트에 활용하기엔, 보안을 위반할 우려가 있음.
- 프로젝트 초기진행 방법
  - LLM/embedding 모델
    - 개인비용으로 유료 api를 사용하여 demo를 진행,
    - OpenAI api 활용

## UI

-Streamlit으로 구현

## 기능

세션들 관리, jump, 그래프 뷰 등

## LLM Model

- Demo: OpenAI - chatgpt-4o-mini  
- 미래: 사내 LLM 모델 api

## RAG

채팅기록 검색

### 사용 DB

- Demo: list + FAISS IVF (구현 단순화 목적)  
- 미래: GraphDB (서비스 특화 성능/속도 최적화 목적)

### 유사도 측정 방식

- Demo: Cosine Similarity / dot product (속도 목적)  
- 미래: + Time diff + Hop count (서비스 특화 최적화 목적)

- Time diff: 대화의 시점이 얼마나 가까운지  
- Hop count: 해당 대화가 대화 graph 상 얼마나 가까운지

### 추출 rule

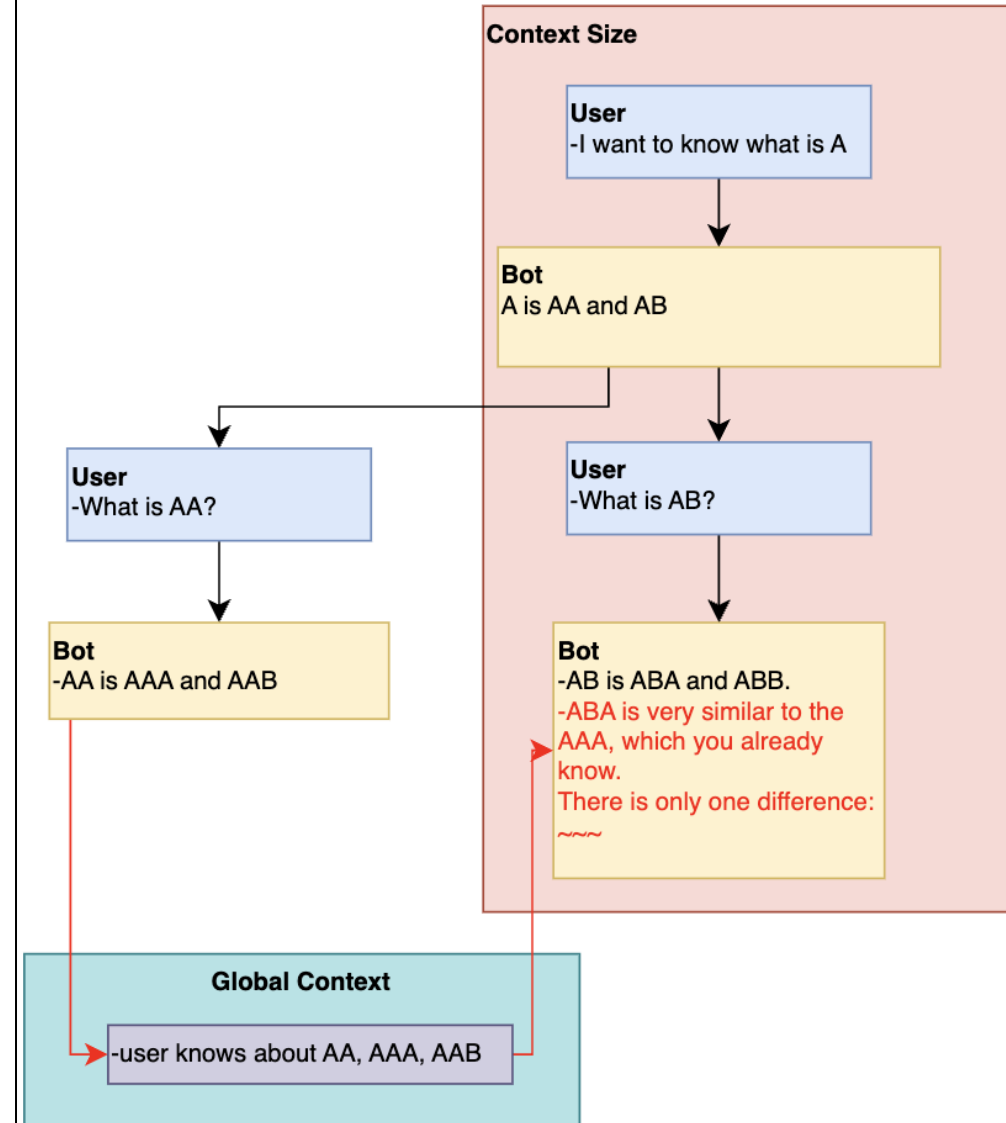
- Demo: top k (빠른 연산 목적)  
- 미래: 계산식으로 점수 측정 후 Threshold 기준  
(모든 경우를 포용하는 정확하고 완벽한 추출 목적)

## Embedding Model

- Demo: OpenAI - text-embedding-3-small  
- 미래: 사내 임베딩 api, or 오픈소스 모델

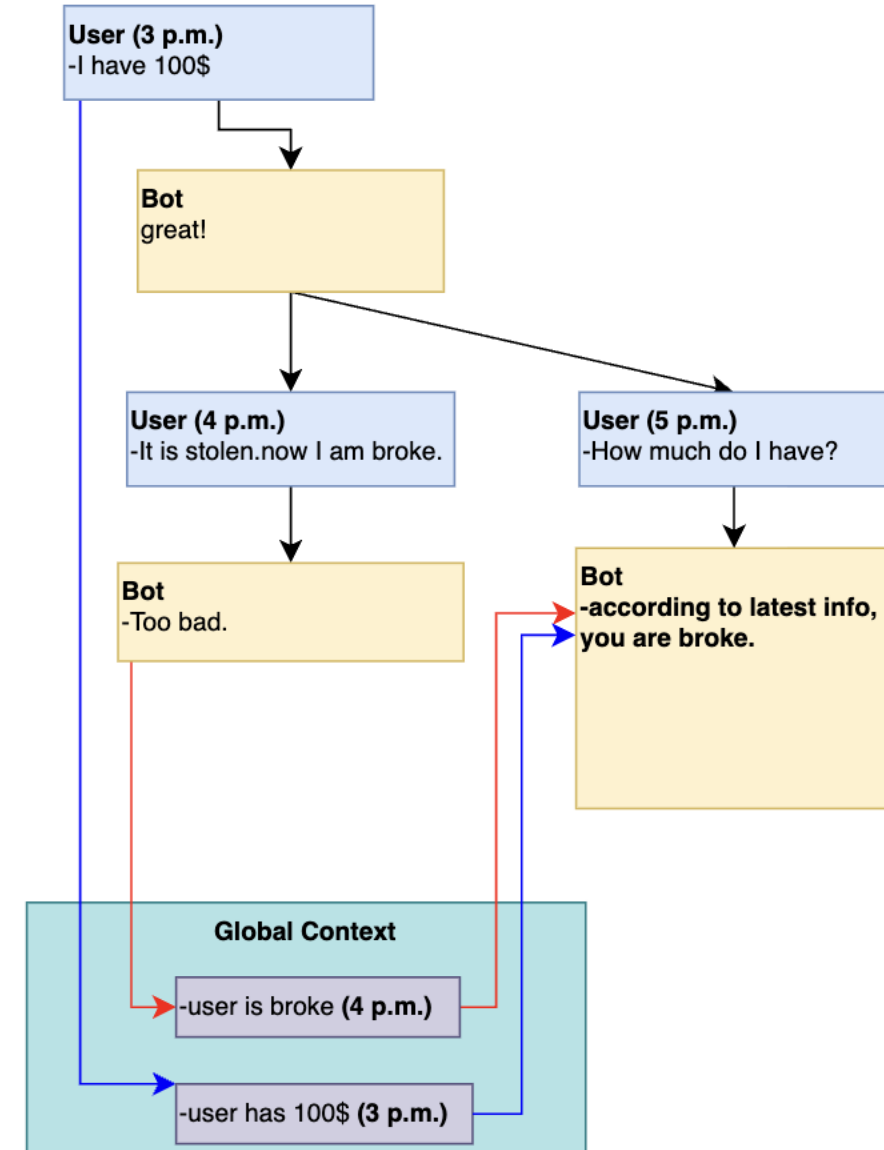
# 추가계획: global context (RAG활용)

- 몇몇 내용은, 다른 branch에서 일어났어도 context에 남는 것이 합리적.
  - 사용자도 반복되는 설명을 피할 수 있음.
- 다만, Global context는 현재와 관련이 없는 값들로 flooding될 가능성 존재
  - RAG로 retrieve되어야 함.
  - outdate될 수 있는 종류는 피해야함.
- “어떤 내용을 저장할지 판단”이 애매하기 때문에,  
대화 node들을 직접 doc으로 관리하는것도 좋은 방법.
- (Global context = 대화노드)



# 추가계획: timestamp 기반 신뢰

- 정보들은 항상 업데이트 될 수 있으며,  
이 때 과거 정보를 참고할 시, 오류 발생.
- RAG로 추출 시에 timestamp도 포함하여,  
최신화된 정보를 기반으로 대화하도록 유도.





구현 중간 milestone

# 성공적으로 gpt-4o-mini로 branched chat 성공

- branch된 대화에서는, 기존 대화의 context가 사용되지 않는 것 확인.
- global context를 제외한 내용 확인 완료.

## • 원본 branch

You: james is my name. remember that

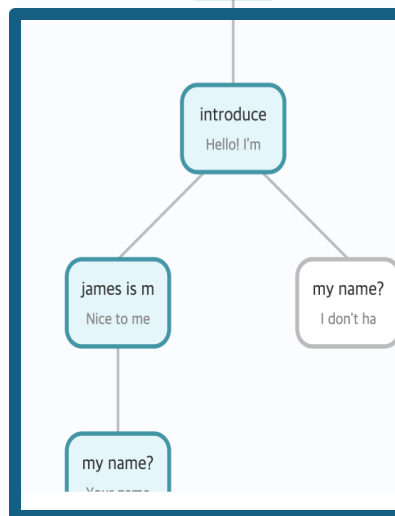
Nice to meet you, James! While I can't remember information from our conversation once it's over, I'll do my best to assist you with anything you need while we're chatting. How can I help you today?

Branch from here

You: my name?

Your name is James! How can I assist you today, James?

Branch from here



## • 신규 branch

You: Hello! What can I help you with today?

Hi! Ask me anything.

Branch from here

You: introduce yourself

Hello! I'm an AI developed to assist you with a wide range of topics, from providing information and answering questions to helping with problem-solving and brainstorming ideas. My goal is to make information accessible and to assist you in the best way possible. How can I help you today?

Branch from here

You: my name?

I don't have access to personal data about users unless it has been shared with me in the course of our conversation. If you'd like to share your name or how you would like me to address you, please let me know!

Branch from here

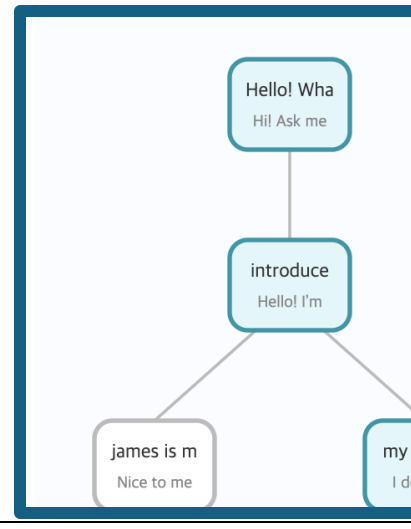
Your message:

my name?

(simulated)

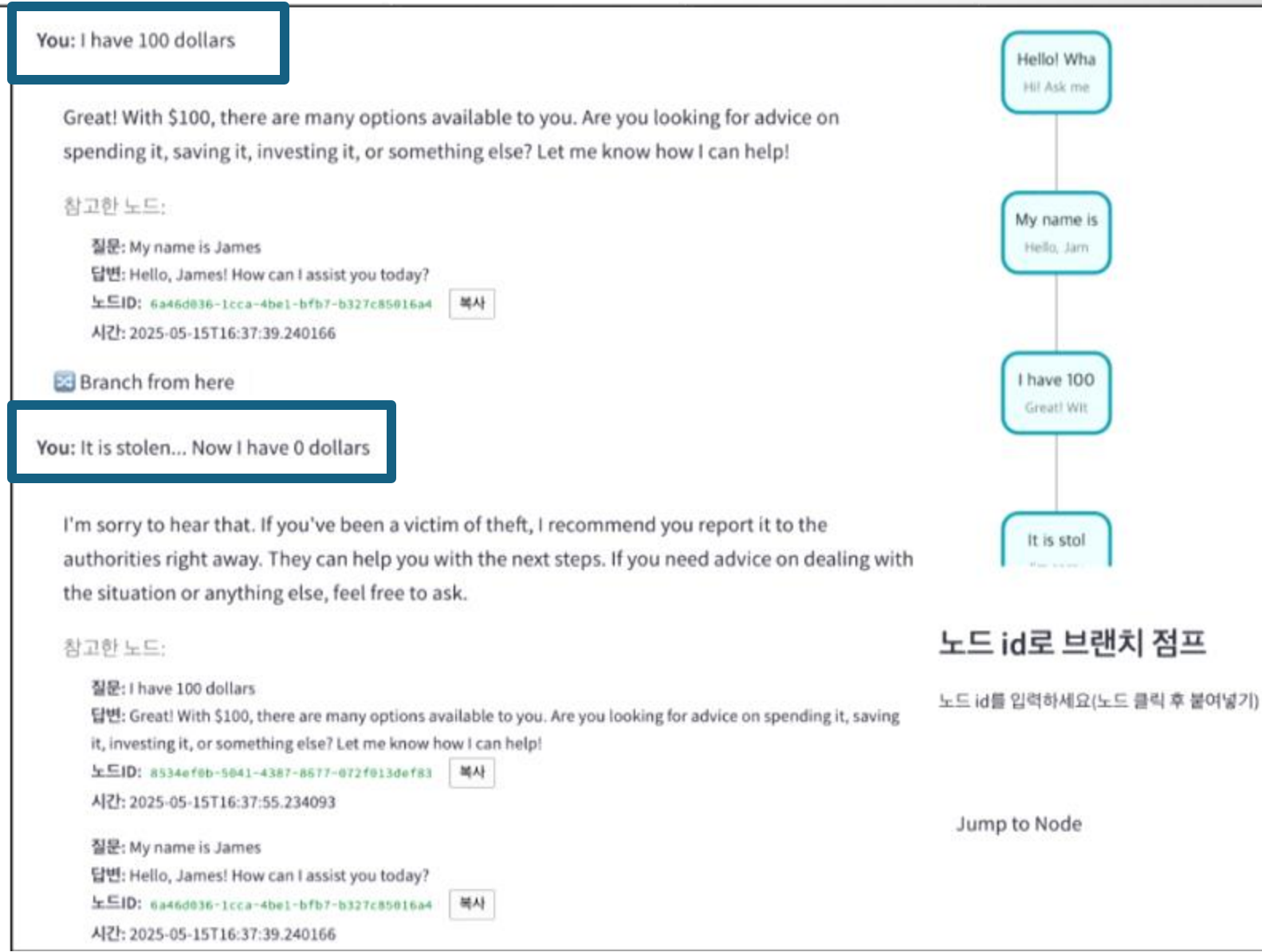
+ Add current input to Global Context

 Conversation Tree View (SVG)



# 성공적으로 RAG로 Global Context 구현 성공 - 1 (Timestamp로 최신정보 구분 성공)

- 테스트 준비:
  - 100달러를 가지고 있다가,
  - 도둑질을 당해 0달러가 되었다고 대화
- 테스트 방법
  - 100달러 소유하고 있는 시점으로 jump
  - 현재 얼마를 가지고 있는지 질문
- 성공 시
  - 0달러
- 실패 시
  - 100달러



# 성공적으로 RAG로 Global Context 구현 성공 - 2

- 두 노드를 모두 참고하였으며, timestamp로 0달러를 최신상태로 구분, 그에 맞는 답변을 한 것을 볼 수 있음.
- FAISS IVF를 사용하여 scalable
  - docs 개수가 쌓이기 전까진 전수조사

You: I have 100 dollars

Great! With \$100, there are many options available to you. Are you looking for advice on spending it, saving it, investing it, or something else? Let me know how I can help!

참고한 노트:

질문: My name is James  
답변: Hello, James! How can I assist you today?  
노트ID: 6a46d836-1cca-4be1-bfb7-b327c85816a4 [복사](#)  
시간: 2025-05-15T16:37:39.240166

[Branch from here](#)

You: How many dollars do I have?

Based on what you've shared, it sounds like you currently have \$0 if your money was stolen. I'm really sorry to hear that. If you need advice on what to do next or how to manage your finances moving forward, feel free to ask!

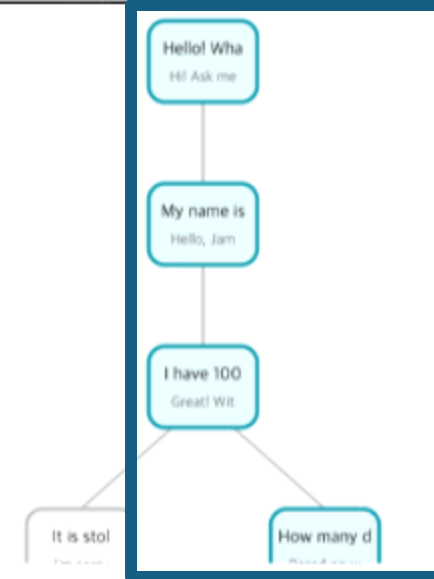
참고한 노트:

질문: I have 100 dollars  
답변: Great! With \$100, there are many options available to you. Are you looking for advice on spending it, saving it, investing it, or something else? Let me know how I can help!  
노트ID: 8534ef0b-5041-4387-8677-072f013def83 [복사](#)  
시간: 2025-05-15T16:37:55.234093

질문: It is stolen... Now I have 0 dollars  
답변: I'm sorry to hear that. If you've been a victim of theft, I recommend you report it to the authorities right away. They can help you with the next steps. If you need advice on dealing with the situation or anything else, feel free to ask.  
노트ID: b81d8aac-92d3-49e2-8cac-92028d56afad [복사](#)  
시간: 2025-05-15T16:38:22.322600

질문: My name is James  
답변: Hello, James! How can I assist you today?  
노트ID: 6a46d836-1cca-4be1-bfb7-b327c85816a4 [복사](#)  
시간: 2025-05-15T16:37:39.240166

[Branch from here](#)



## 노트 id로 브랜치 점프

노트 id를 입력하세요(노트 클릭 후 붙여넣기)

8534ef0b-5041-4387-8677-072f013def83

[Jump to Node](#)

# 편의기능 – Jump Cache

아래 시나리오를 위한 기능:

- 참고한 문서로 Jump 이후 돌아오기
  - 클릭 한번으로 가능
- 자주 돌아가고 싶은 시점 저장
  - 한번 jump 후 고정 가능

## ← 이전 위치로 돌아가기 (Jump History)

이 위치로 돌아가기: i am here to test br | Welcome to BranchCha

고정

삭제

이 위치로 돌아가기: I am Jihoon Kim, SW | Hello, Jihoon Kim! I

고정

삭제

## 💬 Current Conversation

You: Hello! What can I help you with today?

Hi! Ask me anything.

🔗 Branch from here

Your message:

i am here to test branchchat

Send

## 🌳 Conversation Tree View (SVG)

Hello! Wha  
Hi! Ask me

I am Jihoon  
Hello, Jih

i am here  
Welcome to

## 노드 id로 브랜치 점프

노드 id를 입력하세요(노드 클릭 후 붙여넣기)

6bcda57d-423b-4d49-a471-c513a5614ad3

Jump to Node

# 코드구조 설명

# global\_context.py – RAG (FAISS IVF)

- document에는 timestamp가 있으나, 실제 embedding에는 question & answer만 사용함.
- IVF는 최소 문서 양이 필요하여, init 여부를 is\_trained()로 표기

```
def add_document(self, question: str, answer: str, node_id: str, timestamp: Optional[str] = None):
    embedding = self._get_embedding_openai(question + "\n" + answer)
    if embedding is None:
        print(f"Failed to get embedding for document: Q: {question}")
        return None # Or handle error appropriately

    doc = Document(
        question=question,
        answer=answer,
        node_id=node_id,
        timestamp=timestamp or datetime.now().isoformat(),
        embedding=embedding # Storing 1D np.ndarray
    )
    self.documents.append(doc)

    if not self.is_trained:
        if len(self.documents) >= self.min_train_docs:
            self._train_and_populate_index()
        else:
            if self.faiss_index:
                # FAISS expects a 2D array for add, so reshape the 1D embedding
                self.faiss_index.add(embedding.reshape(1, -1))
    return doc
```

# global\_context.py - RAG (FAISS IVF)

- 문서 수가 적을 시에는 brute\_force\_search
  - 모든 doc에 대해 유사도 검사
- 문서 수가 충분할 시 IVF 구조 완성
  - FAISS IVF 사용

```
def search(self, query: str, top_k=3) -> List[Document]:
    if not self.is_trained or not self.faiss_index or self.faiss_index.ntotal == 0:
        # Fallback to brute-force search if IVF not ready and documents exist
        if self.documents:
            print("FAISS index not ready or empty. Falling back to brute-force search.")
            return self._brute_force_search(query, top_k)
        print("FAISS index not trained or empty, and no documents for fallback. Returning no results.")
        return []

    query_emb = self._get_embedding_openai(query)
    if query_emb is None:
        return []

    # nprobe controls the speed-accuracy trade-off for IVF indexes.
    # Higher nprobe = more accurate, but slower.
    self.faiss_index.nprobe = min(self.nlist, 10) # Search up to 10 clusters or all if nlist is small

    # FAISS search expects a 2D array for queries
    similarities, indices = self.faiss_index.search(query_emb.reshape(1, -1), top_k)

    results = []
    if indices.size > 0:
        for i in range(len(indices[0])):
            idx = indices[0][i]
            if idx != -1 and idx < len(self.documents): # Ensure index is valid
                doc = self.documents[idx]
                # Optionally, you can store the similarity score with the document if needed
                # doc.similarity_score = float(similarities[0][i])
                results.append(doc)
    return results
```



# app.py - UI

← 이전 위치로 돌아가기 (Jump History)  
점프 이력이 없습니다.

💬 **Current Conversation**

You: Hello! What can I help you with today?

Hi! Ask me anything.

🔗 Branch from here

Your message:

Send

### 🌳 Conversation Tree View (SVG)

Hello! Wha  
Hi! Ask me

노드 id로 브랜치 점프

노드 id를 입력하세요(노드 클릭 후 붙여넣기)

Jump to Node

### OUTLINE

- global\_context
- ChatNode
  - \_\_init\_\_
  - id
  - parent\_id
  - user\_input
  - response
  - children
- root
- jump\_to\_branch
- render\_chat
- llm\_response
- add\_message
- render\_tree\_svg\_ui
- render\_jump\_history
- col1
- col2
- user\_input

# openai\_chat.py – llm모델 관련

- langchain에 정의된 ChatOpenAI 클래스 사용
- 개인 키를 .env에 입력해서 사용
- 4o-mini로 지정해서 사용

```
openai_chat.py > chat_with_gpt4o
1  from langchain_openai import ChatOpenAI
2  from langchain_core.messages import HumanMessage, SystemMessage, AIMessage
3  from dotenv import load_dotenv
4  import os
5
6  def chat_with_gpt4o(messages, api_key=None, model="gpt-4o-mini"):
7      """
8      messages: list of dicts, e.g. [{"role": "system", "content": "..."}]
9      api_key: OpenAI API Key (None이면 .env 또는 환경변수 사용)
10     model: "gpt-4o" (mini는 gpt-4o로 통합됨)
11     """
12     if api_key is None:
13         load_dotenv()
14         api_key = os.getenv("OPENAI_API_KEY")
15     llm = ChatOpenAI(openai_api_key=api_key, model=model)
16     lc_messages = []
17     for msg in messages:
18         if msg["role"] == "system":
19             lc_messages.append(SystemMessage(content=msg["content"]))
20         elif msg["role"] == "user":
21             lc_messages.append(HumanMessage(content=msg["content"]))
22         elif msg["role"] == "assistant":
23             lc_messages.append(AIMessage(content=msg["content"]))
24     response = llm.invoke(lc_messages)
25     return response.content
26
27 if __name__ == "__main__":
28     # .env 파일에서 키를 불러옵니다.
29     load_dotenv()
30     api_key = os.getenv("OPENAI_API_KEY")
31     messages = [
32         {"role": "system", "content": "You are a helpful assistant."},
33         {"role": "user", "content": "안녕! GPT-4o로 대답해줘."}
34     ]
35     print(chat_with_gpt4o(messages, api_key=api_key))
```

구현 시 겪은 문제점들

# 문제: svg 그림을 직접 streamlit에서 클릭 불가능

- 아래에서 노드 클릭을 해도,
- 화면 변경 안됨



## Conversation Tree View (SVG)



### 해결책

1. `streamlit.components.v1.html`은 return value로 JS 값을 받을 수 없음

→ `streamlit.components.v1.html`은 단방향(파이썬→JS)만 지원,

양방향(클릭→파이썬)은 Custom Components로 직접 만들어야 함.

2. 실제로 JS→파이썬 값을 받으려면

- `streamlit-component-template` 기반의 커스텀 컴포넌트를 만들어야 함
- 또는, `pyvis/streamlit-agraph` 등 이미 양방향 통신이 되는 컴포넌트를 활용

3. 현실적인 대안

- SVG 아래에 노드별 Streamlit 버튼을 두는 방식(이전 방식)으로만 클릭 이벤트를 파이썬에서 받을 수 있음
- 또는, `pyvis/agraph` 등 외부 라이브러리로 전환

### 정리

- SVG 내에서 직접 클릭 → 파이썬 값 전달은 `streamlit.components.v1.html`로는 불가능합니다.
- Streamlit 공식 문서:

> The return value of `st.components.v1.html` is always None.

1

### 추천

1. SVG 아래에 노드별 버튼을 두는 방식으로 돌아가거나,
2. `pyvis/agraph` 등 JS 기반 네트워크 라이브러리로 전환(노드 클릭 지원)

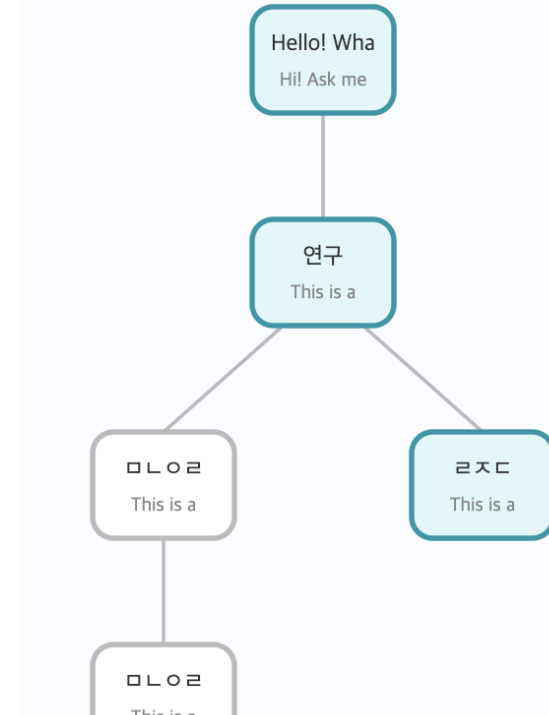
### 어떻게 할까요?

- SVG+Streamlit 버튼 병행(가장 간단, 클릭 확실)
- `pyvis/agraph` 등 외부 라이브러리(진짜 그래프 인터랙션, 클릭 지원)

원하는 방식을 말씀해주시면, 그에 맞춰 바로 구현/가이드드리겠습니다!

# 해결: 노드 id가 복사되도록 변경

- 사용자는 노드 클릭 후,  
복사된 노드 id를 아래 붙여넣기 후 jump 클릭



## 노드 id로 브랜치 점프

노드 id를 입력하세요(노드 클릭 후 붙여넣기)

a44c4427-0a47-44c6-9355-9d480c08e51d

Jump to Node

추후 개선가능성

# 대화내용 저장용량 최적화

- 현재는 session state로 텍스트를 저장하고 있어서,  
마지막에 branching된 두 branch가 전체 대화 context를 따로 가지고 있을 것
- 노드단위로 대화를 node id로 db에 저장,  
session state는 parent node들의 index들만 가지고 있도록 하면  
저장공간 최적화 가능

# 대화내용 Latency 최적화

- 과거 branch로 jump 시 or 현재 branch에서 새로 채팅 invoke 시 attention 과거 토큰들 간의 attention 등 연산을 처음부터 하는 것은 매우 비효율적
- 각 node state가, 해당 시점까지 수행했던 연산들 중, 재사용 가능한 것들은 저장할 필요가 있음.
- 이를 위해서라도, db를 노드 단위로 구성하는 것이 합리적임.
  - leaf node는, 한 단계 위 parent node의 값을 가져와서, 새로 생기는 토큰으로 인해 생기는 연산에 대해서만 추가적인 attention 수행



# RAG 데이터 scope 관리 고도화

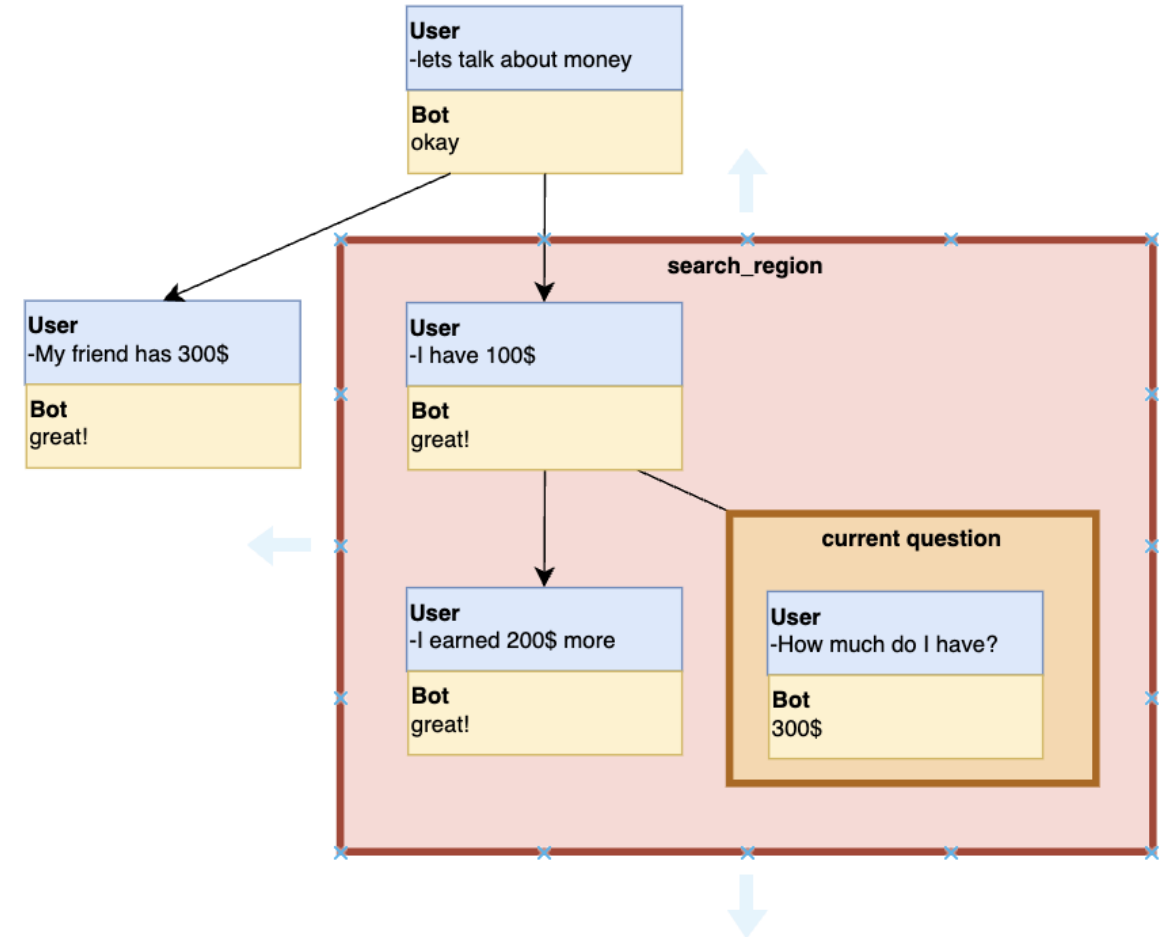
## – 개념

- 특정 대화나 branch는 user 입장에서 장기적으로 기억되고 싶지 않을 수도 있음.
  - 쓸데없는 대화
  - 이미 outdate된, 잘못된 정보에 대한 대화
- 특정 대화는 원하는 node의 child들에서만 rag 데이터로서의 의미가 있을 수도 있음.
  - project specific data
    - chatGPT의 경우, 현재 개발환경을 이야기하면, 질문 시점에서의 일시적 개발환경임에도 personal memory로 저장해버리는 경우가 있다.
- 각 node들이 어느 범위까지 RAG의 유사도 검색 대상으로 쓰일 지를 설정할 수 있게 해야함.
  - LLM으로 자동으로 판단하거나
  - default를 global로 하되, 노드별로 user가 정할 수 있게 한다거나

# RAG 데이터 scope 관리 고도화 개선안 1

## – sub-tree selection

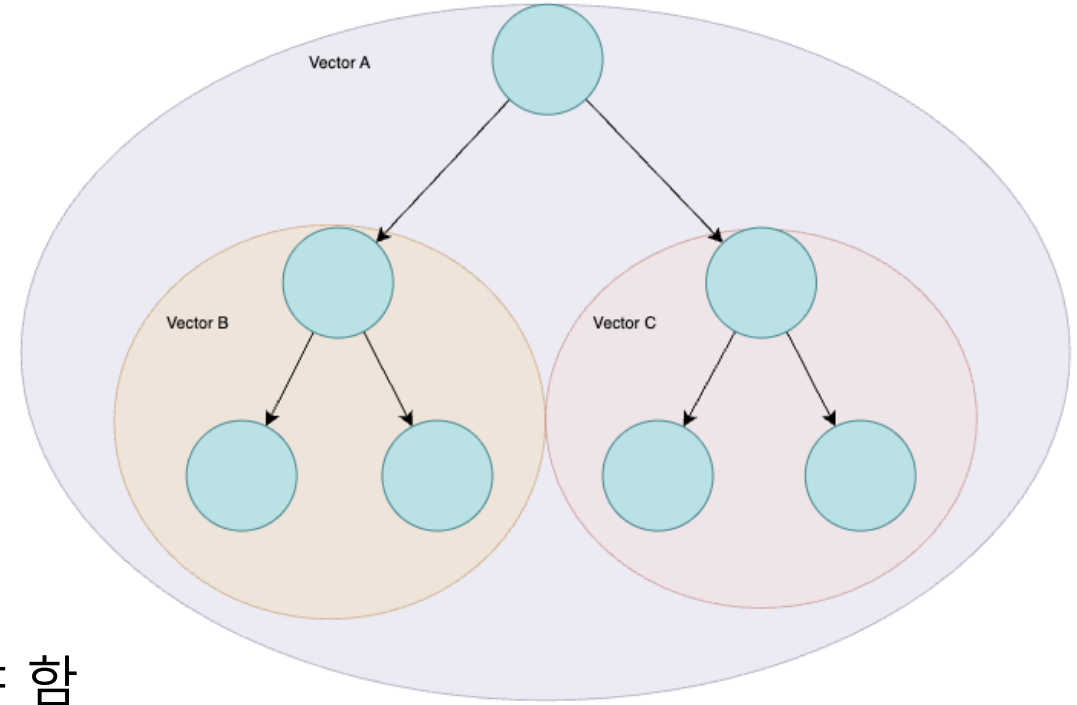
- 어떤 node를 RAG에 포함할 지 결정하는 것을  
user에게 수동으로 맡길 경우, user의 피로도가 올라갈 수 있음.  
user의 workload를 최소화 해야 함.
- 특정 parent node를 지정하면, 해당 tree에서만  
검색하도록 하면, user는 비교적 편안하게 지정 가능  
미지정 시 전체 tree 검색.



# RAG 데이터 scope 관리 고도화 개선안 2

## - node grouping - 개념

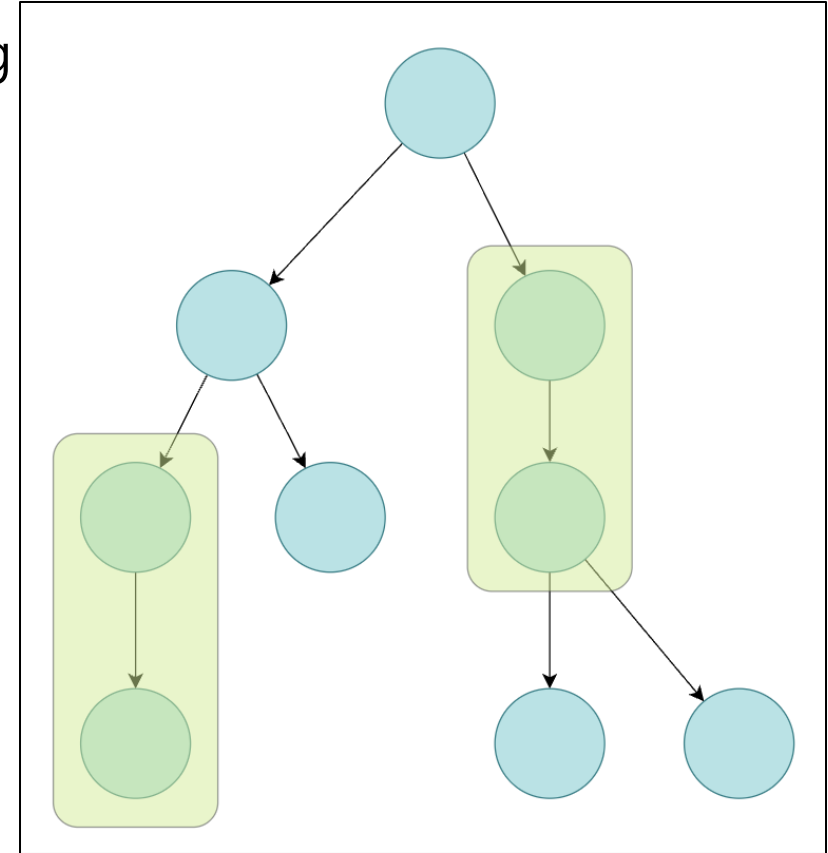
- parent node를 기준으로 group을 생성
  - group을 대표하는 vector 관리
  - group vector 유사도가 낮을 경우 해당 node들을 세부 유사도 계산에서 폐기
- 가정: 한 sub-tree는 비슷한 내용의 대화를 한다.
- 조건1: 길이제한이 여유로운 embedding 모델을 사용해야 함
  - chunk로 나누는 기법을 사용하지 않고 있기 때문
    - ex) BGE-m3, Instructor-XL
- 조건2: group vector update가 쉬워야 함
  - leaf node가 생길 때마다, 모든 ancestor vector를 update해야 하기 때문
    - mean pooling



# RAG 데이터 scope 관리 고도화 개선안 2

## – node grouping – 세부설계1

- grouping의 의미는, 유사도가 떨어지는 node들의 효율적인 pruning
  - branch out되지 않은 multi-turn 대화는 하나의 group으로 간주
  - 유사도가 높은 group에 대해서만, 아래의 구성으로 vector unpack
    - parent node/group
    - child node/groups
- 유사도가 높은 node들이 확보되면 바로 종료
  - top-k, threshold 모두 지원 가능
  - 여러 종류의 모델 지원 가능



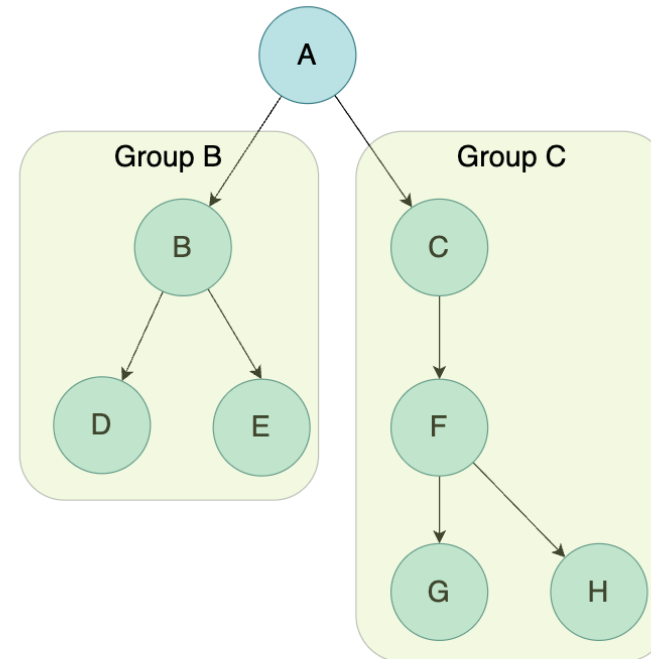
# RAG 데이터 scope 관리 고도화 개선안 2

## – node grouping – 세부설계2

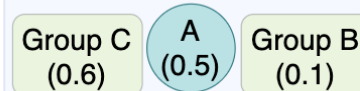
- 8개 노드 top-3 시나리오
  - 6번의 연산만으로 종료

고려 필요한 사항

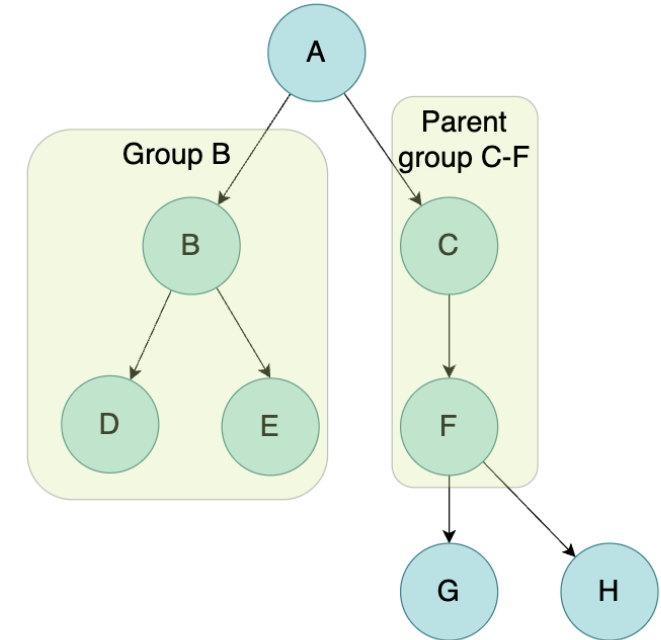
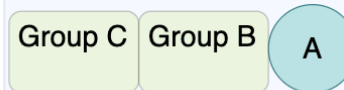
- group 내 outlier 확인 어려움
  - group unpack 조건은 관대하게 설정
  - outlier 포착 여지 상승
- leaf node(대화 추가) 시 vector update
  - 모든 ancestor group vector update 필요
  - mean pooling으로 진행 시 연산부담 하락
  - graph형태의 DB를 사용해야 효율적
  - sub-tree의 sum을 저장해두어야 효율적



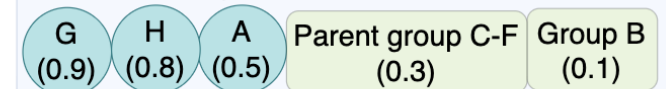
Cosine Similarity Calculation



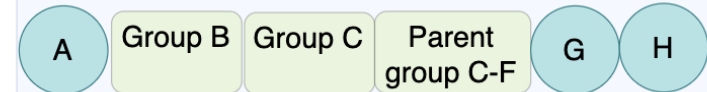
Calculated:



Cosine Similarity Calculation



Calculated:



- 1. 계산 시작 (top-3 인 경우)
- 2. 유사도 높은 group unpack

# RAG 데이터 scope 관리 고도화 개선안 2

## – node grouping – 세부설계3

- scalable하기 위해서는 상용 library를 사용하는 것이 유리하다.
- 이미 비슷한 원리를 사용하려는 IVF (Inverted File Indexing) 기법이 존재한다.
  - 벡터를 clustering해서, 가까울 확률이 높은 노드들부터 확인하여 연산 최소화
- 물론, 실제 chat graph를 활용 하는 것의 장점도 있겠으나,
- FAISS IVF로 문제가 생길 시 고려할 문제로 보인다.
- 실제 구현: Brute Force & FAISS IVF
  - FAISS IVF는 기본 클러스터링 학습을 위해 최소 document의 숫자가 필요
  - 그 전까지는 Brute Force로 모든 vector를 계산하도록 해둔 상태

기타 서비스 스크린샷

## Current Conversation

**You:** Hello! What can I help you with today?

Hi! Ask me anything.

 Branch from here

**You:** I am Jihoon Kim, SW engineer

Hello, Jihoon Kim! It's great to meet you. As a software engineer, I'm sure you're involved in exciting projects.  
How can I assist you today?

 Branch from here

**You:** i am here to test branchchat

Welcome to BranchChat! If you have any questions or if there's something specific you'd like to test or discuss, feel free to let me know. I'm here to help!

참고한 노드:

질문: I am Jihoon Kim, SW engineer

답변: Hello, Jihoon Kim! It's great to meet you. As a software engineer, I'm sure you're involved in exciting projects. How can I assist you today?

노드ID:

aba2d7a0-32b5-4953-80bf-632dcdf83e7a

위의 Copy 버튼을 눌러 클립보드에 복사하세요.

시간: 2025-05-16T14:12:34.150713

 Branch from here

Hello! Wha  
Hi! Ask me

I am Jihoon  
Hello, Jih

i am here  
Welcome to

## 노드 id로 브랜치 점프

노드 id를 입력하세요(노드 클릭 후 붙여넣기)

Jump to Node



## ← 이전 위치로 돌아가기 (Jump History)

이 위치로 돌아가기: AA is AAA and AAB | It looks like you're

고정

삭제

## 🗨️ Current Conversation

**You:** Hello! What can I help you with today?

Hi! Ask me anything.

🔗 Branch from here

**You:** A is AA and AB

It seems like you're providing information about blood types. Blood type "A" can be either AA (homozygous) or AB (heterozygous).

- **AA** means that both alleles inherited are of type A.
- **AB** means one allele is A and the other is B.

If you have a specific question or need more information about blood types, genetics, or related topics, please let me know!

🔗 Branch from here

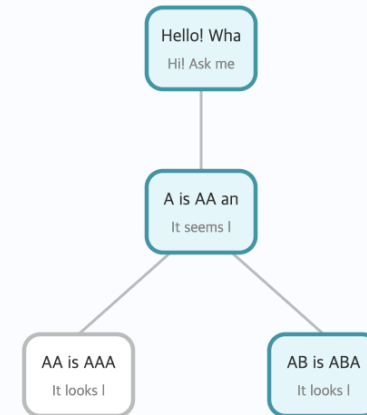
**You:** AB is ABA and ABB

It looks like you are continuing the pattern of allele combinations for blood types. Let's clarify the breakdown:

If we take AB as a reference:

- **ABA** would imply one A allele is inherited twice (the second A and B imply that there is a mix of A and B alleles, similar to AB genotype).
- **ABB** implies two B alleles and one A allele. In terms of standard blood types, this combination would not typically occur because blood type is determined by two alleles.

## 🌳 Conversation Tree View (SVG)



## 노드 id로 브랜치 점프

노드 id를 입력하세요(노드 클릭 후 붙여넣기)

Jump to Node