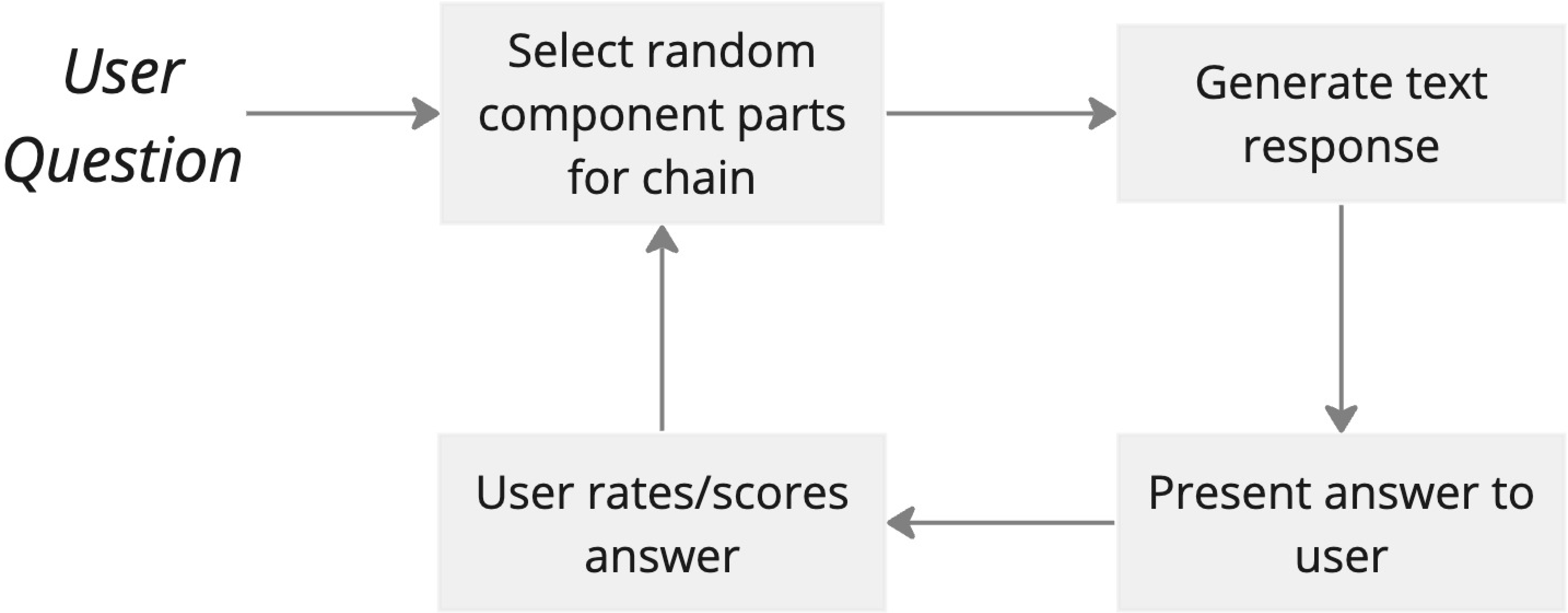
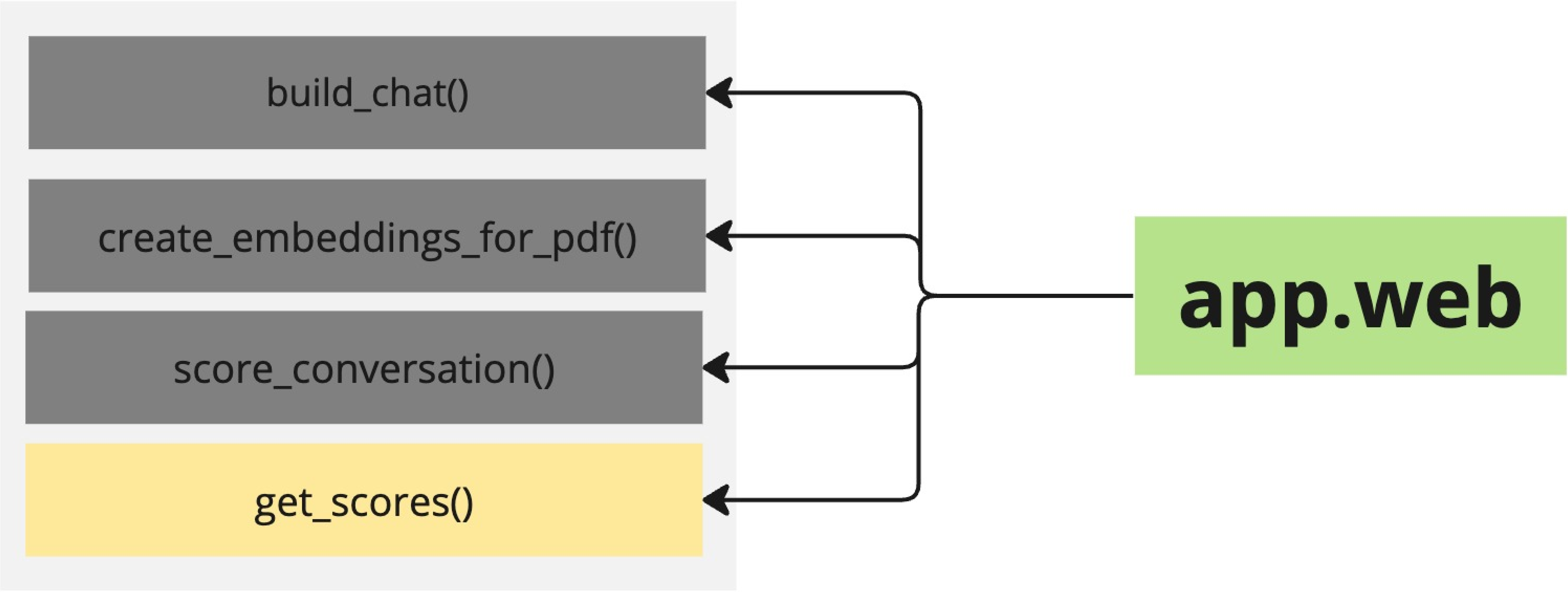

User Question
Select random component parts for chain
Generate text response
Present answer to user
User rates/scores answer

app.chat Module



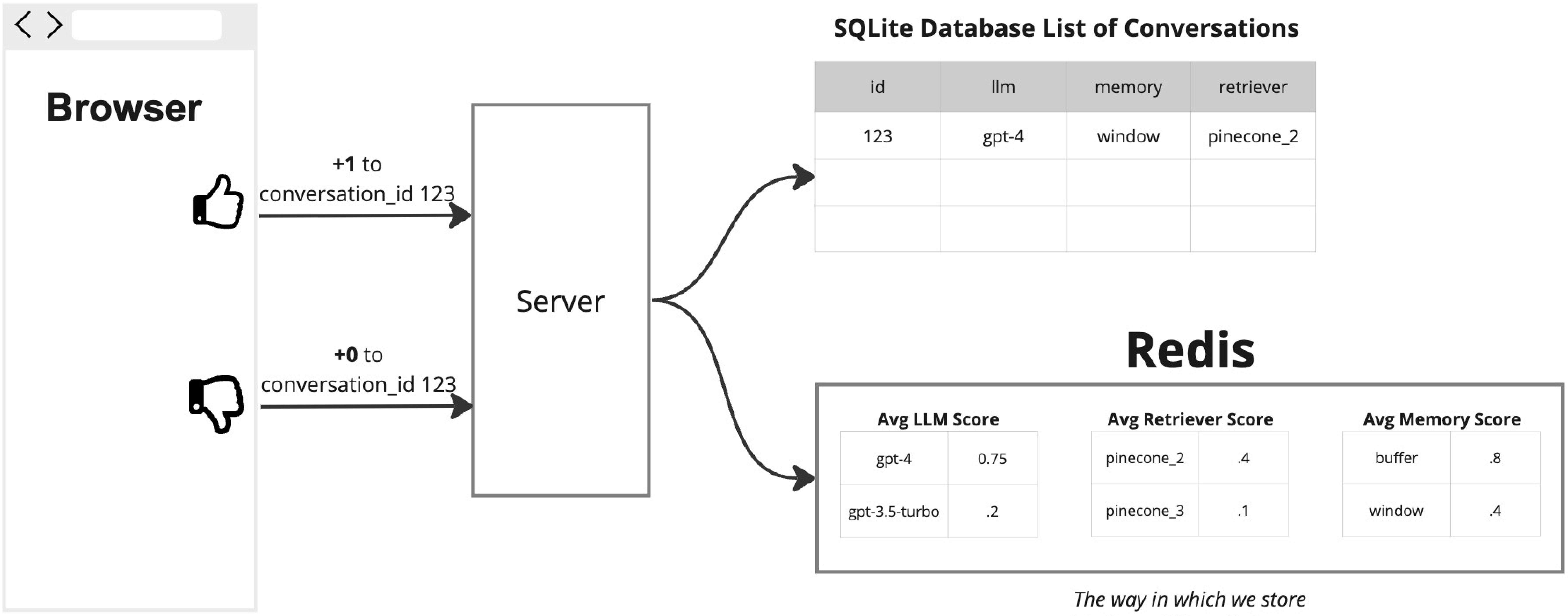
build_chat()

create_embeddings_for_pdf()

score_conversation()

get_scores()

app.web



SQLite Database List of Conversations

id	llm	memory	retriever
123	gpt-4	window	pinecone_2

Redis

Avg LLM Score

gpt-4	0.75
gpt-3.5-turbo	.2

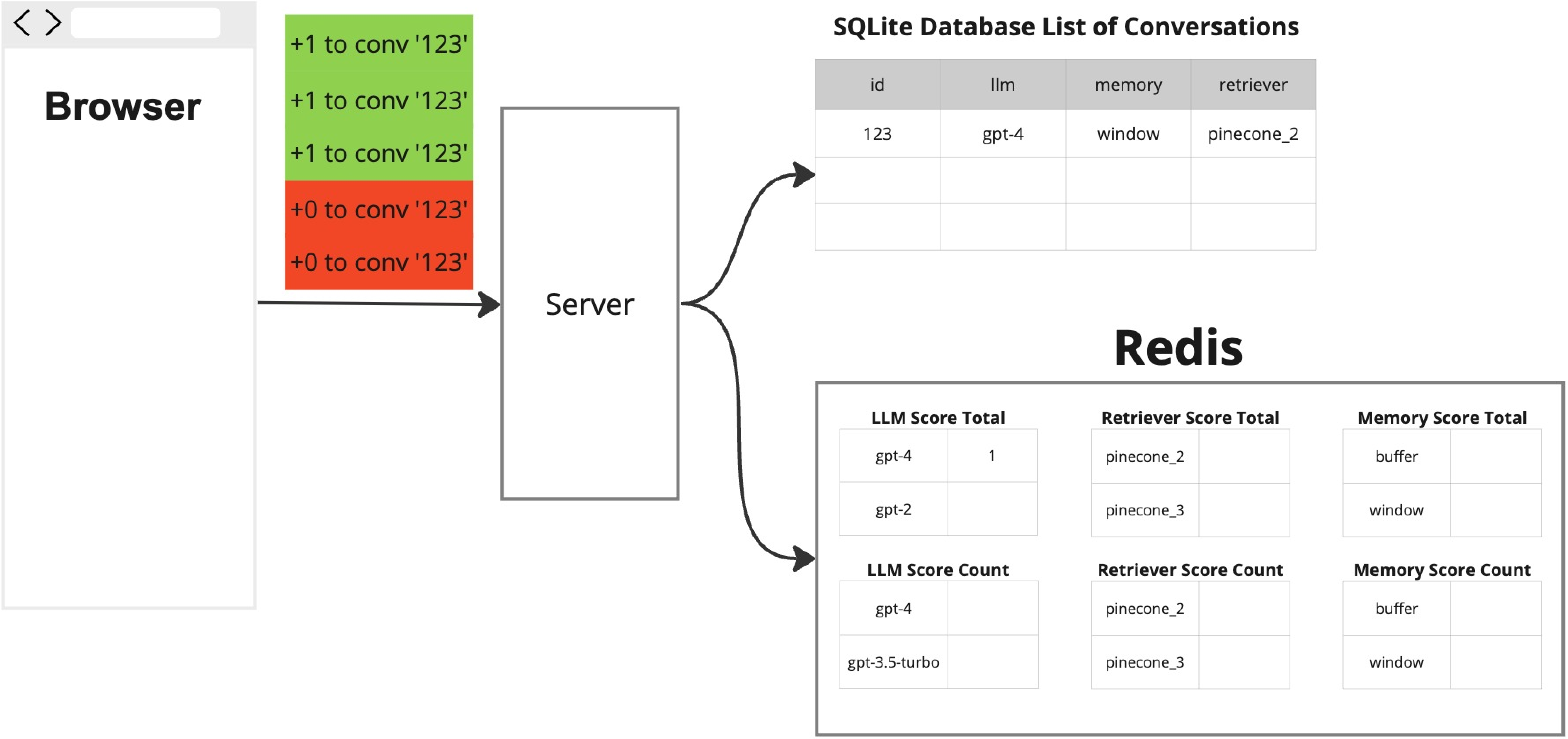
Avg Retriever Score

pinecone_2	.4
pinecone_3	.1

Avg Memory Score

buffer	.8
window	.4

The way in which we store this data will be a little more complicated



SQLite Database List of Conversations

id	llm	memory	retriever
123	gpt-4	window	pinecone_2

Redis

LLM Score Total

gpt-4	1
gpt-2	

Retriever Score Total

pinecone_2	
pinecone_3	

Memory Score Total

buffer	
window	

LLM Score Count

gpt-4	
gpt-3.5-turbo	

Retriever Score Count

pinecone_2	
pinecone_3	

Memory Score Count

buffer	
window	

Redis

LLM Score Total

gpt-4	1
gpt-3.5-turbo	5
gpt-2	1

LLM Score Count

gpt-4	10
gpt-3.5-turbo	5
gpt-2	3

LLM Map

gpt-4	*builder*
gpt-3.5-turbo	*builder*

Get all the values and counts from Redis

Take all the keys from the LLM map (avoid calculating average for old/removed LLM's)

For each active LLM, calculate average score. Use min of .1 to make sure a LLM always has a chance of being used

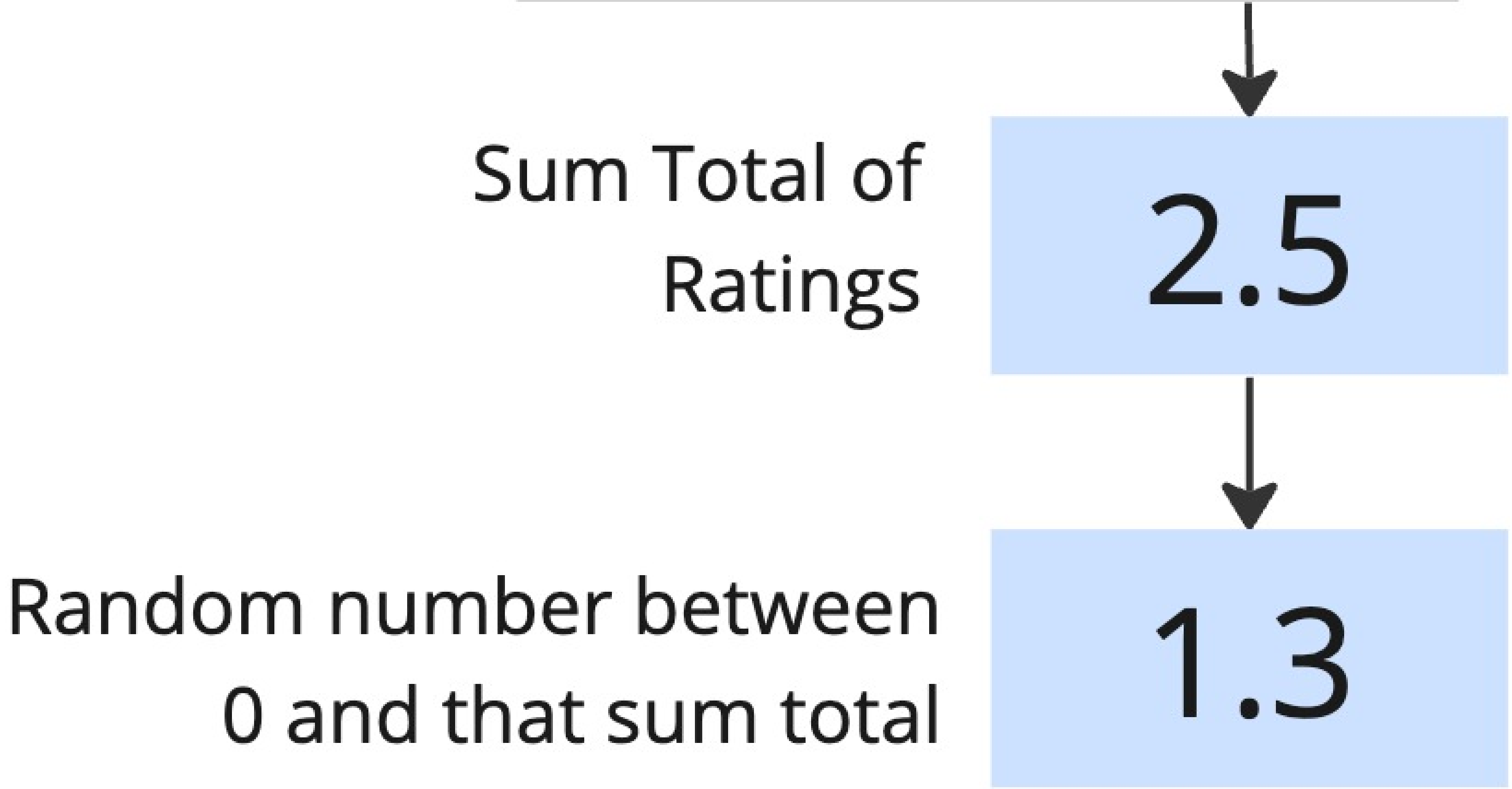
Calculate sum total of all ratings

Pick a random number between 0 and the sum total

Use the random number + sum total to pick a random component

Average Scores

pinecone_1	0.5
pinecone_2	1.0
pinecone_3	1.0

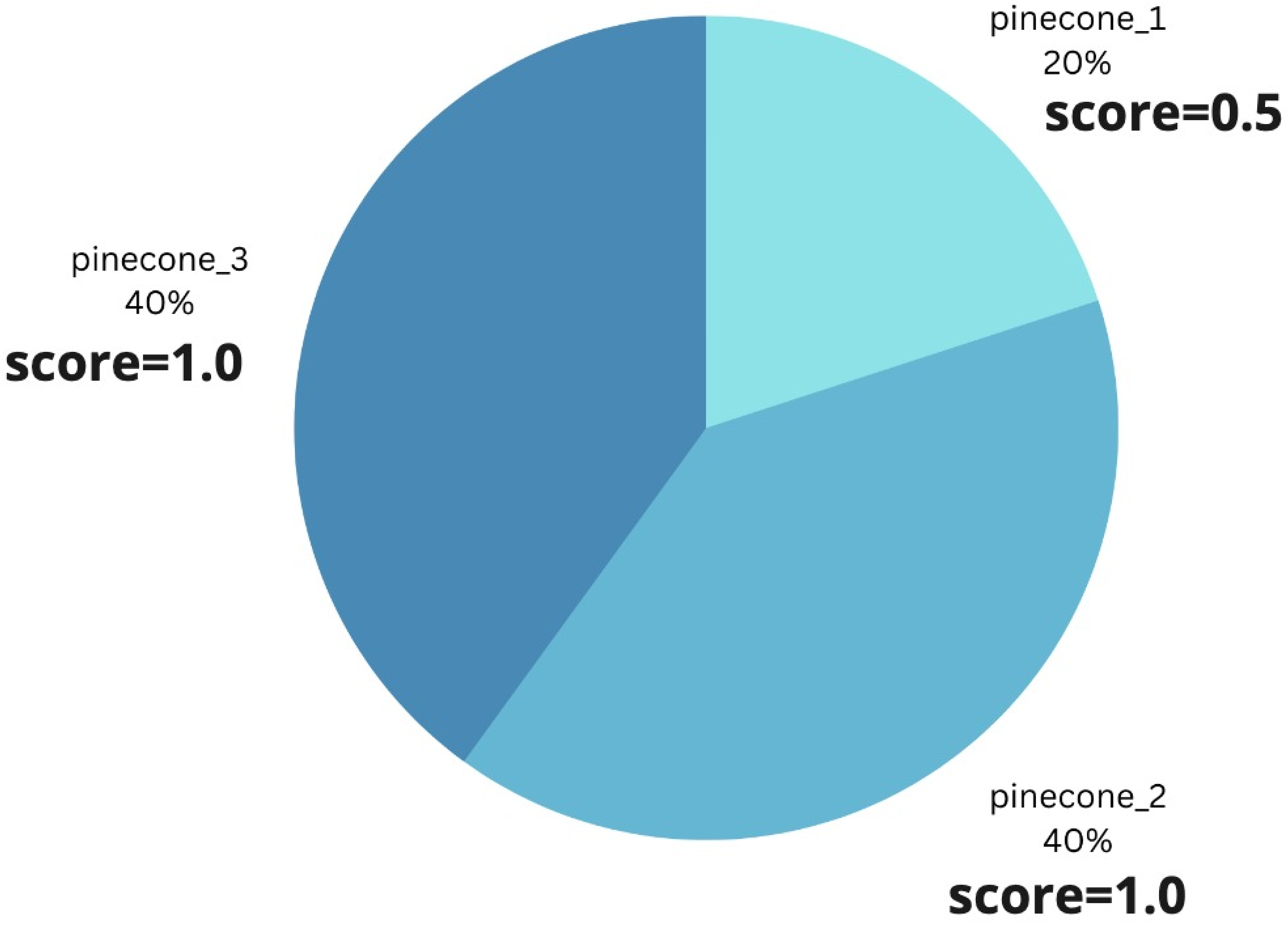


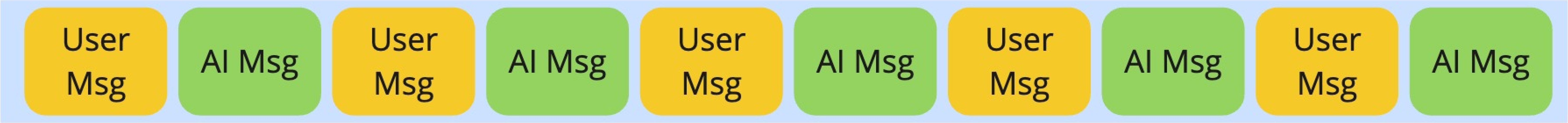
Sum Total of Ratings

2.5

Random number between 0 and that sum total

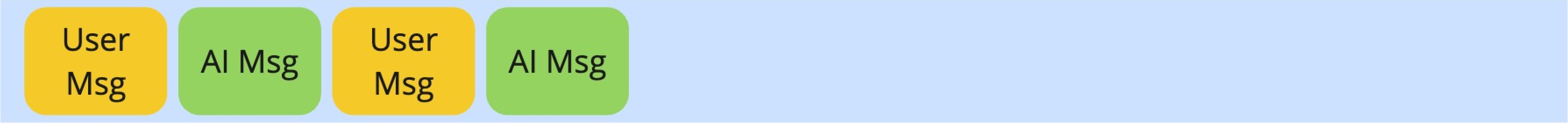
1.3


pinecone_1
20%
score=0.5
pinecone_3
40%
score=1.0
pinecone_2
40%
score=1.0



ConversationBufferMemory

Stores all messages of
the conversation



ConversationBufferWindowMemory

Stores only the last **K=2**
exchanges