GENERATIVE AI COURSE
ASSIGNMENT - 1

Nikki Gautam

Strategies for Managing Context Limitations and Reducing Noise in Conversational AI

- ## Understanding the Problem:

Modern conversational AI systems, (like ChatGPT) or virtual assistants, rely on a context window - a limit on how much information the model can "remember" during a conversation.

This is measured in tokens, and exceeding the token limit means earlier parts of the conversation get forgotten.

This leads to two main challenges:

1. **Context Loss** - Important earlier messages are cut off
2. **Noise Accumulation** - Irrelevant or outdated information is retained which causes confusion

As conversational systems become more complex and multi-turn interactions increase, it becomes essential to manage the context window efficiently and reduce noise to ensure responses remain accurate and coherent.

- **Techniques Explored (their Pros and Cons):**

1. Summarisation of Chat History:

Summarising earlier parts of the conversation into a compact format

- **Types**:
    - Full Summarisation
    - Rolling (partial) summarisation
    - Hierarchical summarisation

PROS	CONS
Keeps essence of old messages	May miss finer details
Reduces token load significantly	Requires summarisation logic or API calls
Mimics human memory well	Quality depends on summarisation model

2. Windowed Context / Recent Message Retention:

Only keeping the last N user-bot message pairs

PROS	CONS
Very lightweight	Forgets long-term context completely
Easy to implement	Cannot handle call-backs to earlier queries

3. Retrieval-Augmented Generation (RAG)

Using vector embeddings to fetch only relevant past content from a database or knowledge base

PROS	CONS
High relevance and context awareness	Requires a retrieval system setup
Scales well to long histories	Needs chunking and scoring logic

4. Token/Chunk Limit Control

Defining limits on number of tokens per chunk and number of chunks retrieved

PROS	CONS
Maintains token budget	Might cut off useful content
Optimizes performance	Needs testing and tuning

5. Embedding-Based Relevance Filtering

Comparing embeddings of new query Vs historical context to keep only relevant ones

PROS	CONS
High semantic precision	Slightly slower (requires computation)
Reduces noise drastically	Needs embedding model and vector search

- **Code Snippet (LangChain)** (Example)**:**

Using LangChain's ConversationSummaryBufferMemory:

```
from langchain.memory import ConversationSummaryBufferMemory
from langchain.chat_models import ChatOpenAI
from langchain.chains import ConversationChain

llm = ChatOpenAI(temperature=0)
memory = ConversationSummaryBufferMemory(llm=llm, max_token_limit=100)

conversation = ConversationChain(
    llm=llm,
    memory=memory,
    verbose=True
)

conversation.predict(input="Hi! My name is Aisha.")
conversation.predict(input="Can you remember what I told you?")
conversation.predict(input="Let's talk about AI now.")
```

The setup has the following features:

- Keeps recent messages intact
- Summarises older messages when token limit is reached

One can also use **ConversationBufferWindowMemory** to retain only the last ***k*** messages

- **Comparing the Outcomes of the Study:**

Strategy	Context Retention	Token Efficiency	Relevance	Ease of Use
Full Summarisation	Medium	High	Medium	Medium
Rolling Summarisation	High	High	High	Medium
Windowed Context	Low	Very High	Low	Very Easy
RAG (Retrieval-Augmented)	Very High	High	Very High	Medium Hard
Embedding Filtering	High	Medium	High	Medium

- **Conclusion (Which Strategy Works the Best):**

Based on the research and practical experimentation, the most effective strategy is:

A hybrid approach combining Rolling Summarisation and Retrieval-Augmented Generation (RAG)

This ensures that:

- Old messages are retained meaningfully through summaries
- Relevant context is retrieved using semantic similarity
- Token limits are respected, maintaining performance

This approach offers a balanced trade-off between performance, coherence, and memory cost, especially in long multi-turn conversations.