

RELATÓRIO TÉCNICO - (UNIDADE 4)

Disciplina: Inteligência Artificial - Turma 01 (2025.2)

Professor: Hendrik Macedo

Instituição: Universidade Federal de Sergipe (UFS) / Departamento de Computação (DCOMP)

Integrantes:

Ellen Karolliny dos Santos

Ellen Vitoria Menezes Lima

João Santos Rocha

Larissa Batista dos Santos

Tasso Marcel de Oliveira

1. Definição do Problema

O problema abordado foi a manutenção do histórico de conversa em implementações de chat puro, sem utilização de banco de dados. Em vários sistemas conversacionais baseados em Modelos de Linguagem de Grande Escala (LLMs), o histórico de conversa não é armazenado por padrão. O modelo não “lembra” interações anteriores de forma automática, apenas formula uma resposta com base na pergunta atual. Sendo assim, o desafio do projeto foi preservar o contexto da conversa para garantir coerência entre perguntas e respostas.

2. Solução do Problema

Foi implementado um mecanismo de memória a nível de aplicação, utilizando:

- Estrutura de dados do tipo lista (*history*)
- Identificação de papéis (*system, user, assistant*)
- Reconstrução manual do prompt a cada requisição
- Função de truncamento para controle de tamanho

A cada nova pergunta do usuário, ela é adicionada ao histórico. Após isso, o histórico é convertido em texto sequencial e o prompt completo é enviado ao modelo. Por fim, a resposta é adicionada ao histórico.

O modelo de linguagem utilizado possui a capacidade de compreender o contexto acumulado, manter coerência semântica, continuar raciocínios anteriores e gerar respostas de acordo com o contexto.

3. Correspondência com o pseudocódigo:

3.1 - A função do agente conversacional:

```
function CONVERSATIONAL-AGENT(user_input, history, config) returns a response, updated_history

return reply_text, updated_history

51 # Agente Conversacional
52 def CONVERSATIONAL_AGENT(user_input, history, config):
53     history.append({"role": "user", "content": user_input})
54     payload = TRUNCATE_HISTORY(history, config["max_tokens"])
55
56     api_response = CALL_LLM_API(config, payload)
57
58     reply_text = api_response
59     history.append({"role": "assistant", "content": reply_text})
60
61     return reply_text, history
62
```

Essa função recebe a pergunta do usuário, o histórico da conversa e as configurações do modelo. Ela retorna a resposta da pergunta e atualiza o histórico de conversa.

3.1 - Estrutura de configuração (hiperparâmetros do modelo):

```
7     raise ValueError('GEMINI_API_KEY não enc
8
9     client = genai.Client(api_key=api_key)
10
11 #Configuração do Modelo
12 MODEL_CONFIG = {
13     "model_name": "gemini-2.5-flash",
14     "temperature": 0.7,
15     "max_tokens": 5000
16 }
17
18 # Gestão de Contexto
19 def TRUNCATE_HISTORY(history, limit):
```

structure MODEL-CONFIG
provider string (e.g., "openai", "local-llama")
model_name string (e.g., "gpt-4o", "dall-e-3", "whisper-1")
temperature float [0.0, 2.0] // Criatividade
max_tokens integer
modality string { "text", "image", "audio" }

3.2 - Construção da Mensagem

```

65 # Parte visual
66 st.title("Chatbot Gemini")
67 st.caption("Modelo: gemini-2.5-flash")
68
69 if "history" not in st.session_state:
70     st.session_state.history = [
71         {"role": "system", "content": "Você é um programador Python."}
72     ]
73
74 for msg in st.session_state.history:
75     if msg["role"] == "user":
76         with st.chat_message("user"):
77             st.markdown(msg["content"])
78     elif msg["role"] == "assistant":
79         with st.chat_message("assistant"):
80             st.markdown(msg["content"])
81
82 user_input = st.chat_input("Digite uma pergunta ... ")
83
84 if user_input:
85     with st.chat_message("user"):
86         st.markdown(user_input)
87

```

2. Cenário A: Texto (Chatbot com memória)

Este pseudocódigo foca na manutenção do **histórico de conversa**, que é implementações de chat puro sem banco de dados.

```

function CONVERSATIONAL-AGENT(user_input, history, config) returns :
    // 1. Construção da Mensagem
    current_msg CREATE-MESSAGE(role="user", content=user_input)
    APPEND(history, current_msg)

```

```

    // 2. Gestão de Contexto (Obrigatório para não estourar tokens)
    // Mantém o System Prompt fixo e remove mensagens antigas se necessário
    payload TRUNCATE-HISTORY(history, limit=config.max_tokens)

```

3.3 - Gestão de contexto (função que faz a gestão do contexto para não estourar o limite de tokens):

```

9 client = genai.Client(api_key=api_key)
10
11 #Configuração do Modelo
12 MODEL_CONFIG = {
13     "model_name": "gemini-2.5-flash",
14     "temperature": 0.7,
15     "max_tokens": 5000
16 }
17
18 # Gestão de Contexto
19 def TRUNCATE_HISTORY(history, limit):
20     if len(history) > limit:
21         # Mantém o System Prompt fixo e remove
22         # history.pop(1)
23     return history
24

```

implementações de chat puro sem banco de dados.

```

function CONVERSATIONAL-AGENT(user_input, history, config) returns :
    // 1. Construção da Mensagem
    current_msg CREATE-MESSAGE(role="user", content=user_input)
    APPEND(history, current_msg)

```

```

    // 2. Gestão de Contexto (Obrigatório para não estourar tokens)
    // Mantém o System Prompt fixo e remove mensagens antigas se necessário
    payload TRUNCATE-HISTORY(history, limit=config.max_tokens)

```

```

    // 3. Chamada de Inferência (Texto)
    api_response CALL-LLM-API(config, payload)

```

3.4 - Chamada de Inferência

```

1 def CALL_LLM_API(config, payload):
2     # Constrói o prompt a partir do histórico
3     contents = []
4     for msg in payload:
5         if msg["role"] == "user":
6             contents.append({
7                 "role": "user",
8                 "parts": [{"text": msg["content"]}])
9
10        elif msg["role"] == "assistant":
11            contents.append({
12                "role": "model",
13                "parts": [{"text": msg["content"]}])
14
15
16        response = client.models.generate_content(
17            model_config["model_name"],
18            contents=contents,
19            config={
20                "temperature": config["temperature"],
21                "max_output_tokens": config["max_tokens"]
22            }
23
24
25        return response.text

```

2. Cenário A: Texto (Chatbot com memória)

Este pseudocódigo foca na manutenção do **histórico de conversa**, que é implementações de chat puro sem banco de dados.

```

function CONVERSATIONAL-AGENT(user_input, history, config) returns :
    // 1. Construção da Mensagem
    current_msg CREATE-MESSAGE(role="user", content=user_input)
    APPEND(history, current_msg)

```

```

    // 2. Gestão de Contexto (Obrigatório para não estourar tokens)
    // Mantém o System Prompt fixo e remove mensagens antigas se necessário
    payload TRUNCATE-HISTORY(history, limit=config.max_tokens)

```

```

    // 3. Chamada de Inferência (Texto)
    api_response CALL-LLM-API(config, payload)

```

3.4 - Pós processamento (envio da pergunta do usuário ao chat)

```
81
82 user_input = st.chat_input("Digite uma pergunta ... ")
83
84 if user_input:
85     with st.chat_message("user"):
86         st.markdown(user_input)
87
88     with st.spinner("Pensando ... "):
89         try:
90             resposta, st.session_state.history = CONVERSATIONAL_AGENT(
91                 user_input,
92                 st.session_state.history,
93                 MODEL_CONFIG
94             )
95
96             with st.chat_message("assistant"):
97                 st.markdown(resposta)
98
99         except Exception as e:
100             st.error(f"Erro de Conexão: {e}")
```

// 4. Pós-processamento
reply_text EXTRACT-CONTENT(api_response)
reply_msg CREATE-MESSAGE(role="assistant", content=reply_text)
APPEND(updated_history, reply_msg)