**🏗️ Arsitektur & Alur Kerja Agent**

**Arsitektur Sistem**

graph TD

**Komponen Utama**

**1. LLM Engine**

* **Model**: Google Gemini 1.5 Flash
* **Temperature**: 0.2 (konsisten, tidak terlalu kreatif)
* **API**: Google AI Studio API

llm = ChatGoogleGenerativeAI(

model="gemini-1.5-flash",

temperature=0.2

)

**2. Agent Executor**

* **Type**: Conversational ReAct Description
* **Framework**: LangChain Agents
* **Verbose**: True (untuk debugging)

agent = initialize\_agent(

tools=TOOLS,

llm=llm,

agent=AgentType.CONVERSATIONAL\_REACT\_DESCRIPTION,

memory=memory,

verbose=True

)

**3. Tools System**

Available tools:

* get\_current\_time: Mendapatkan waktu saat ini
* say\_hello: Menyapa user dengan nama

**4. Memory System**

* **Storage**: SQLite Database
* **Type**: Episodic Memory
* **Persistence**: Cross-session memory

**Data Flow Diagram**

User Input → Agent → Memory Retrieval → LLM Analysis → Tool Selection → Execution → Response → Memory Storage → Output

**Keunggulan Arsitektur**

**✅ Modular Design**

* Tools dapat ditambah/dihapus dengan mudah
* Memory system terpisah dan configurable
* LLM engine dapat diganti tanpa mengubah core logic

**✅ Persistent Memory**

* SQLite database untuk long-term storage
* Session-based conversation tracking
* Cross-restart memory retention

**✅ Error Resilience**

* Comprehensive error handling
* Graceful degradation
* User-friendly error messages

**✅ Scalable**

* Easy to add new tools
* Configurable memory backends
* Multiple LLM support potential

**Limitasi Arsitektur**

**❌ Single User**

* Saat ini hanya support satu user session
* Tidak ada user authentication

**❌ Limited Tools**

* Hanya 2 tools basic
* Tidak ada web search atau advanced capabilities

**❌ No RAG**

* Tidak ada knowledge base integration
* Tidak ada document retrieval system

**Technical Stack**

* **Python**: 3.13+
* **LangChain**: 0.3.26
* **Google Gemini**: 1.5 Flash model
* **SQLite**: Database untuk persistent memory
* **python-dotenv**: Environment variable management

**Pendekatan Memory System**

**Overview Memory Architecture**

Agent menggunakan **Episodic Memory** approach dengan SQLite sebagai backend storage.

**Jenis Memory: Episodic Memory**

**✅ Mengapa Episodic Memory?**

**1. Sequential Conversation Storage**

history = SQLChatMessageHistory(

session\_id="default",

connection="sqlite:///memory.db"

)

**2. Temporal Context Preservation**

* Menyimpan urutan percakapan chronologically
* Mempertahankan context antar interactions
* Enable follow-up questions dan references

**3. Session-based Organization**

* Setiap conversation memiliki session ID
* Mudah untuk isolate atau group conversations
* Potential untuk multi-user implementation

**Memory Implementation Details**

*# Memory Configuration*

memory = ConversationBufferMemory(

chat\_memory=history,

memory\_key="chat\_history",

input\_key="input",

return\_messages=True

)

**Key Components:**

* **SQLChatMessageHistory**: Persistent storage layer
* **ConversationBufferMemory**: In-memory buffer untuk active session
* **session\_id**: Identifier untuk conversation tracking

**Memory Retrieval Process**

**1. Session Initialization**

*# Load existing history on startup*

history = SQLChatMessageHistory(session\_id="default", ...)

**2. Context Loading**

*# Automatic context injection*

memory.load\_memory\_variables({})

*# Returns: {'chat\_history': [HumanMessage, AIMessage, ...]}*

**3. Memory Update**

*# Automatic after each interaction*

memory.save\_context(

{"input": user\_input},

{"output": agent\_response}

)

**Memory Performance**

**✅ Advantages of Current Approach**

**1. Fast Retrieval**

* SQLite queries are very fast
* No embedding computation needed
* Immediate context loading

**2. Simple Implementation**

* No complex vector operations
* Standard SQL operations
* Easy to debug and maintain

**3. Cost Effective**

* No embedding model costs
* Minimal computational overhead
* Standard database operations

**4. Reliable Persistence**

* SQLite is battle-tested
* ACID compliance
* No data loss risk

**❌ Limitations**

**1. Context Window Constraints**

*# Limited by LLM context window*

*# Cannot retrieve very old conversations efficiently*

**2. No Semantic Search**

* Cannot find related topics from past
* No conceptual similarity matching
* Linear search through history only

**3. Scalability Issues**

* Performance degrades with very long conversations
* Memory buffer size limitations
* No intelligent conversation summarization

**Memory Optimization Strategies**

**Current Optimizations:**

**1. Buffer Management**

*# ConversationBufferMemory manages active context*

memory = ConversationBufferMemory(

memory\_key="chat\_history",

return\_messages=True,

max\_token\_limit=2000 *# Prevent context overflow*

)

**2. Session Isolation**

*# Separate sessions prevent memory pollution*

history = SQLChatMessageHistory(session\_id="user\_123")

**Future Optimization Ideas:**

**1. Memory Summarization**

*# Potential implementation*

memory = ConversationSummaryBufferMemory(

llm=llm,

max\_token\_limit=2000,

return\_messages=True

)

**2. Hybrid Memory System**

*# Combine episodic + semantic*

class HybridMemory:

def \_\_init\_\_(self):

self.episodic = SQLChatMessageHistory()

self.semantic = VectorStore() *# For concepts*

**3. Smart Context Selection**

*# Retrieve relevant context based on current query*

def get\_relevant\_context(query, history, max\_items=10):

*# Smart selection instead of just recent messages*

pass

**Memory Monitoring**

**Current Metrics:**

* Database size: memory.db file size
* Active context length: Number of messages in buffer
* Query response time: SQLite query performance

**Monitoring Implementation:**

*# Monitor memory usage*

def check\_memory\_stats():

db\_size = os.path.getsize("memory.db")

context\_length = len(memory.chat\_memory.messages)

print(f"DB Size: {db\_size} bytes, Context: {context\_length} messages")

**Conclusion**

Episodic memory approach memberikan:

* ✅ **Simplicity**: Easy to implement dan maintain
* ✅ **Reliability**: Persistent storage dengan SQLite
* ✅ **Performance**: Fast retrieval untuk recent context
* ✅ **Cost-effectiveness**: No embedding costs

Suitable untuk:

* Personal assistant applications
* Conversational AI dengan context awareness
* Applications dengan moderate conversation length
* Prototyping dan development phase