```
!pip install -qU gradio langchain-cohere faiss-cpu python-dotenv langchain-community
```

```
import os
from langchain_core.example_selectors import SemanticSimilarityExampleSelector
from langchain_core.prompts import FewShotPromptTemplate, PromptTemplate
from langchain_community.vectorstores import FAISS
```

## Config and Params

```
MODEL_TEMPRETURE = 0.1
TOP_N_MATCH = 2 # Tune it as per number of examples in input data

# Load fron .env file
os.environ["COHERE_API_KEY"] = "xxxx"

DEFAULT_ANSWER = '"Sorry, I can not provide answer to given question."'

SYSTEM_PROMPT = f"""You are an enterprise grader customer support agent for Thoughtful AI.

# TASK
- You are provided some examples and a user question.
- You need to answer user question only as per given example match.

# RULES
- If user question matches any example, provide the answer as it is mentioned in the example word by word.
- If user question doesn't match any example, provide MANDATORILY {DEFAULT_ANSWER}.
- Never answer any user question which does not match with provided examples.
- If there is no match, only say {DEFAULT_ANSWER}.
"""
```

```
Start coding or generate with AI.
```

## Loading Models

```
from langchain_cohere import CohereEmbeddings, ChatCohere
```

```
if "COHERE_API_KEY" not in os.environ:
    raise ValueError("COHERE_API_KEY environment variable not set. Please set your Google API key.")

embeddings = CohereEmbeddings(
    model="embed-english-v3.0",
    # input_type="search_query"  # Optimized for question answering
)

llm = ChatCohere(
    model="command-r",
    temperature=MODEL_TEMPRETURE,
    max_tokens=1024
)
```

## Raw Data

```
thoughtful_ai_qa = {
    "questions": [
        {
            "question": "What does the eligibility verification agent (EVA) do?",
            "answer": "EVA automates the process of verifying a patient’s eligibility and benefits information in real-time, eliminating
        },
        {
            "question": "What does the claims processing agent (CAM) do?",
            "answer": "CAM streamlines the submission and management of claims, improving accuracy, reducing manual intervention, and ac
        },
        {
            "question": "How does the payment posting agent (PHIL) work?",
            "answer": "PHIL automates the posting of payments to patient accounts, ensuring fast, accurate reconciliation of payments ar
        },
        {
            "question": "Tell me about Thoughtful AI's Agents.",
            "answer": "Thoughtful AI provides a suite of AI-powered automation agents designed to streamline healthcare processes. These
        },
        {
            "question": "What are the benefits of using Thoughtful AI's agents?",
            "answer": "Using Thoughtful AI's Agents can significantly reduce administrative costs, improve operational efficiency, and r
        }
    ]
}


examples = [
    {"input": q["question"], "output": q["answer"]}
    for q in thoughtful_ai_qa["questions"]
]

print(f"Number of examples: {len(examples)}")
examples[0]
```

```
Number of examples: 5
{'input': 'What does the eligibility verification agent (EVA) do?',
 'output': 'EVA automates the process of verifying a patient’s eligibility and benefits information in real-time, eliminating
manual data entry errors and reducing claim rejections.'}
```

## Vectorization

### Key Implementation Details:

1. **Vector Similarity Search**:
    - Uses `SemanticSimilarityExampleSelector` with FAISS vector store
    - Embeds questions using OpenAI's text-embedding-3-small model
    - Retrieves top 3 similar questions
2. **Few-Shot Prompting**:
    - Combines retrieved examples with the user's question
    - Uses structured prompt template to guide the LLM

Start coding or generate with AI.

```
example_selector = SemanticSimilarityExampleSelector.from_examples(
    examples,
    embeddings,
    FAISS,
    k=TOP_N_MATCH  # Retrieve top N similar examples
)



# %% Create few-shot prompt template
example_prompt = PromptTemplate(
    input_variables=["input", "output"],
    template="Question: {input}\nAnswer: {output}"
)
```

## Retrieve Top-N

```
few_shot_prompt = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    prefix="You are a customer support agent for Thoughtful AI. Answer questions using these examples:",
    suffix="Question: {input}\nAnswer:",
    input_variables=["input"]
)
```

## Chatbot Agent

### 1. **Semantic Retrieval from Predefined Dataset**

- The agent uses semantic similarity search to retrieve the most relevant answers from a hardcoded dataset about Thoughtful AI.
- By leveraging vector embeddings and FAISS, it ensures that user queries are matched to the closest available knowledge, even if the wording differs.

2. **Few-Shot Prompt Construction**

- For each user question, the agent dynamically constructs a prompt using the top-matching examples from the dataset.
- This few-shot approach improves answer accuracy and relevance, as the LLM is guided by real, contextually similar Q&A pairs.

3. **LLM Fallback for Out-of-Domain Queries**

- If the agent cannot find any relevant examples in the dataset, it automatically falls back to a generic LLM response.
- This ensures that the chatbot remains conversational and helpful, even for queries outside the predefined scope.

4. **Robust Error Handling**

- The function is wrapped in a try-except block to catch and report any unexpected errors gracefully.
- Handles missing or malformed data in the examples, skipping problematic entries without interrupting the user experience.
- If no relevant examples are found, the agent still provides a meaningful response rather than failing silently.

5. **Maintainability and Extensibility**

- The function is modular and easy to update: new Q&A pairs can be added to the dataset without code changes.
- The agent logic is separated from the UI layer, making it adaptable for CLI, web, or API-based interfaces.

6. **Session History Support**

- The function signature includes a `history` parameter, allowing for future enhancements such as context-aware multi-turn conversations.

```
# Main Chat Agent
def chatbot_ai_agent(message: str, history: list = None) -> str:
    try:
        # Retrieve similar examples (modified to handle empty results)
        similar_examples = example_selector.select_examples({"input": message}) or []

        if not similar_examples:
            return llm.invoke(f"Answer about Thoughtful AI: {message}").content

        # Format examples with proper error handling
        formatted_examples = []
        for ex in similar_examples:
            try:
                formatted_examples.append(example_prompt.format(**ex))
            except KeyError:
                continue

        # Build final prompt with fallback
        final_prompt = f"""{SYSTEM_PROMPT}.

        # Examples:
        {''.join(formatted_examples)}

        # User question:
        {message}
        Answer:"""

        return llm.invoke(final_prompt).content

    except Exception as e:
        return f"Sorry, Something went wrong..!!"
```

## Testing

```
# Case-1: Exact Match
chatbot_ai_agent(message="What does the eligibility verification agent (EVA) do?")
```

```
'EVA automates the process of verifying a patient’s eligibility and benefits information in real-time, eliminating manual data entr
```

```
# Case-2: Semantic Match
chatbot_ai_agent(message="What does EVA do?")
```

```
'EVA automates the process of verifying a patient’s eligibility and benefits information in real-time, eliminating manual data entr
```

```
# Case-3: No Match
chatbot_ai_agent(message="what is capital of USA?")
```

```
'Sorry, I can not provide answer to given question.'
```

## Chatbot

```
import gradio as gr


# Gradio interface with validated parameters
demo = gr.ChatInterface(
    fn=chatbot_ai_agent,
    title="Thoughtful AI Support Agent",
    description="Ask about Thoughtful AI's products and solutions",
    examples=[q["question"] for q in thoughtful_ai_qa["questions"]],
    cache_examples=True,
    autofocus=True,
    multimodal=False  # Explicitly disable multimodal input
)
```

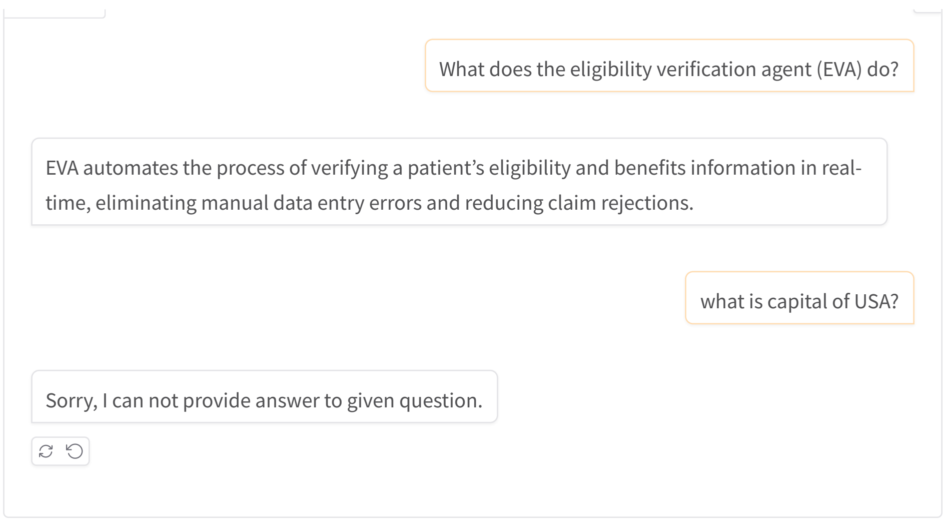
```
/usr/local/lib/python3.11/dist-packages/gradio/chat_interface.py:338: UserWarning: The 'tuples' format for chatbot messages is depre
  self.chatbot = Chatbot(
```

Start coding or generate with AI.

```
# Final launch
demo.launch(
    share=True,
    inline=True,
    show_error=True,
    debug=False  # Disable debug mode for production
)
```

```
Caching examples at: '/content/.gradio/cached_examples/17'
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://66636bdafe9831883e.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```



What does the eligibility verification agent (EVA) do?

EVA automates the process of verifying a patient’s eligibility and benefits information in real-time, eliminating manual data entry errors and reducing claim rejections.

what is capital of USA?

Sorry, I can not provide answer to given question.

Start coding or generate with AI.