```python
import os
import json
from langchain_openai import AzureOpenAIEmbeddings, AzureChatOpenAI
from langchain_community.vectorstores import FAISS

AZURE_OPENAI_ENDPOINT = os.environ["AZURE_OPENAI_ENDPOINT"]
AZURE_OPENAI_API_KEY = os.environ["AZURE_OPENAI_API_KEY"]

embeddings = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    api_key=AZURE_OPENAI_API_KEY,
    model="text-embedding-ada-002",
    api_version="2023-05-15"
)
vectorstore = FAISS.load_local("faiss_kb", embeddings, allow_dangerous_deserialization=True)

llm = AzureChatOpenAI(
    openai_api_key=AZURE_OPENAI_API_KEY,
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    openai_api_version="2025-01-01-preview",
    model="gpt-4o-mini",
    temperature=0
)

def retrieve_kb(user_question, top_k=5):
    docs_and_scores = vectorstore.similarity_search_with_score(user_question, k=top_k)
    hits = []
    for doc, score in docs_and_scores:
        hits.append({
            "doc_id": doc.metadata["doc_id"],
            "answer_snippet": doc.page_content,
            "source": doc.metadata["source"]
        })
    return hits

def generate_answer(user_question, kb_hits):
    kb_str = "\n".join([f"[{hit['doc_id']}] {hit['answer_snippet']}" for hit in kb_hits])
    prompt = (
        "You are a software best-practices assistant.\n"
        "User Question:\n"
        f"{user_question}\n\n"
        "Retrieved Snippets:\n"
        f"{kb_str}\n\n"
        "Task:\nBased on these snippets, write a concise answer to the userвЂ™s question.\n"
        "Cite each snippet you use by its doc_id in square brackets (e.g., [KB004]).\n"
        "Return only the answer text."
    )
    messages = [
        {"role": "system", "content": "You are a software best-practices assistant."},
        {"role": "user", "content": prompt}
    ]
    response = llm.invoke(messages)
    content = getattr(response, "content", None)
    if not content and hasattr(response, "choices"):
        content = response.choices[0].message.content
    return content.strip()

def critique_answer(user_question, initial_answer, kb_hits):
    kb_str = "\n".join([f"[{hit['doc_id']}] {hit['answer_snippet']}" for hit in kb_hits])
    prompt = (
        f"You are a critical QA assistant. The user asked: {user_question}\n\n"
        f"Initial Answer:\n{initial_answer}\n\n"
        f"KB Snippets:\n{kb_str}\n\n"
        "Task:\nDetermine if the initial answer fully addresses the question using only these snippets.\n"
        "- If it does, respond exactly: COMPLETE\n"
        "- If it misses any point or cites missing info, respond: REFINE: <short list of missing topic keywords>\n\n"
        "Return exactly one line."
    )
    messages = [
        {"role": "system", "content": "You are a critical QA assistant."},
        {"role": "user", "content": prompt}
    ]
    response = llm.invoke(messages)
    content = getattr(response, "content", None)
    if not content and hasattr(response, "choices"):
        content = response.choices[0].message.content
    return content.strip()

def refine_answer(user_question, initial_answer, critique_result, kb_hits):
    missing_keywords = critique_result.replace("REFINE:", "").strip()
    new_query = f"{user_question} and information on {missing_keywords}"
    extra_snippet = retrieve_kb(new_query, top_k=1)
    snippet_str = f"[{extra_snippet[0]['doc_id']}] {extra_snippet[0]['answer_snippet']}" if extra_snippet else ""
    prompt = (
        f"You are a software best-practices assistant refining your answer. The user asked: {user_question}\n\n"
        f"Initial Answer:\n{initial_answer}\n\n"
        f"Critique: {critique_result}\n\n"
        f"Additional Snippet:\n{snippet_str}\n\n"
        "Task:\nIncorporate this snippet into the answer, covering the missing points.\n"
        "Cite any snippet you use by doc_id in square brackets.\n"
        "Return only the final refined answer."
    )
    messages = [
        {"role": "system", "content": "You are a software best-practices assistant."},
        {"role": "user", "content": prompt}
    ]
    response = llm.invoke(messages)
    content = getattr(response, "content", None)
    if not content and hasattr(response, "choices"):
        content = response.choices[0].message.content
    return content.strip()

def answer_user_question(user_question):
    kb_hits = retrieve_kb(user_question, top_k=5)
    print("\nKB Hits:")
    for hit in kb_hits:
        print(f"[{hit['doc_id']}] {hit['answer_snippet']}")
    initial_answer = generate_answer(user_question, kb_hits)
    print("\nInitial Answer:\n", initial_answer)
    critique_result = critique_answer(user_question, initial_answer, kb_hits)
    print("\nCritique Result:", critique_result)
    if critique_result.startswith("COMPLETE"):
        final_answer = initial_answer
    elif critique_result.startswith("REFINE"):
        refined = refine_answer(user_question, initial_answer, critique_result, kb_hits)
        print("\nRefined Answer:\n", refined)
        final_answer = refined
    else:
        final_answer = initial_answer
    print("\nFinal Output:\n", json.dumps({"answer": final_answer}, indent=2))
    return {"answer": final_answer}

if __name__ == "__main__":
```

```
sample_questions = [
    "What are best practices for caching?",
    "How should I set up CI/CD pipelines?",
    "What are performance tuning tips?",
    "How do I version my APIs?",
    "What should I consider for error handling?"
]
for q in sample_questions:
    print("\n" + "="*60)
    print("User Question:", q)
    answer_user_question(q)
```