I want to use TiDB for rag in our agent,

Example 1 code:

This example demonstrates how to use PyTiDB to build a minimal RAG application.

- Use Ollama to deploy local embedding model and LLM model
- Use Streamlit to build a Web UI for the RAG application
- Use PyTiDB to build a minimal RAG application

main.py:

```python
import os
import dotenv

import litellm
from litellm import completion
import streamlit as st

from pytidb import TiDBClient
from pytidb.schema import TableModel, Field
from pytidb.embeddings import EmbeddingFunction

dotenv.load_dotenv()
litellm.drop_params = True


# Define the embedding and LLM models
EMBEDDING_MODEL = "ollama/mxbai-embed-large"
LLM_MODEL = "ollama/gemma3:4b"
PROMPT_TEMPLATE = """Context information is below.
---------------------
{context}
---------------------
Given the information and not prior knowledge, answer the query
in a detailed and precise manner.

Query: {question}
Answer:"""


def stream_text_only(llm_response):
    for chunk in llm_response:
        yield chunk.choices[0].delta.content or ""


# Connect to TiDB
```

```
db = TiDBClient.connect(
    host=os.getenv("TIDB_HOST", "localhost"),
    port=int(os.getenv("TIDB_PORT", "4000")),
    username=os.getenv("TIDB_USERNAME", "root"),
    password=os.getenv("TIDB_PASSWORD", ""),
    database=os.getenv("TIDB_DATABASE", "rag_example"),
    ensure_db=True,
)


# Define the Chunk table
text_embed = EmbeddingFunction(EMBEDDING_MODEL)


class Chunk(TableModel):
    __tablename__ = "chunks"
    # Notice: Avoid table already defined error when streamlit rerun the
script.
    __table_args__ = {"extend_existing": True}

    id: int = Field(primary_key=True)
    text: str = Field()
    text_vec: list[float] = text_embed.VectorField(
        source_field="text",
    )


table = db.create_table(schema=Chunk, if_exists="skip")


# Insert sample chunks
sample_chunks = [
    "Llamas are camelids known for their soft fur and use as pack
animals.",
    "Python's GIL ensures only one thread executes bytecode at a time.",
    "TiDB is a distributed SQL database for HTAP and AI workloads.",
    "Einstein's theory of relativity revolutionized modern physics.",
    "The Great Wall of China stretches over 13,000 miles.",
    "Ollama enables local deployment of large language models.",
    "HTTP/3 uses QUIC protocol for improved web performance.",
    "Kubernetes orchestrates containerized applications across
clusters.",
    "Blockchain technology enables decentralized transaction systems.",
    "GPT-4 demonstrates remarkable few-shot learning capabilities.",
    "Machine learning algorithms improve with more training data.",
    "Quantum computing uses qubits instead of traditional bits.",
```

```python
    "Neural networks are inspired by the human brain's structure.",
    "Docker containers package applications with their dependencies.",
    "Cloud computing provides on-demand computing resources.",
    "Artificial intelligence aims to mimic human cognitive functions.",
    "Cybersecurity protects systems from digital attacks.",
    "Big data analytics extracts insights from large datasets.",
    "Internet of Things connects everyday objects to the internet.",
    "Augmented reality overlays digital content on the real world.",
]

if table.rows() == 0:
    chunks = [Chunk(text=text) for text in sample_chunks]
    table.bulk_insert(chunks)

# Initialize chat history
if "messages" not in st.session_state:
    st.session_state.messages = []

# Display chat messages from history on app rerun
if len(st.session_state.messages) > 0:
    for message in st.session_state.messages:
        st.chat_message(message["role"]).markdown(message["content"])
else:
    with st.chat_message("assistant"):
        st.markdown(
            """
        👋 Hello! I'm your AI assistant powered by TiDB.

        Try asking me:
        - What's TiDB?
        - How to deploy LLM locally?
        """
        )

# React to user input
final_response = ""
if user_question := st.chat_input("Say something ..."):
    st.chat_message("user").markdown(user_question)

    # Add user message to chat history
    messages = st.session_state.messages.copy()
    st.session_state.messages.append(
        {
            "role": "user",
            "content": user_question,
        }
    )

    final_response = ""
    with st.chat_message("assistant") as assistant_message:
        # Retrieve (R): search relevant chunks with user message.
        with st.status("Retrieving relevant documents...") as status:
            res =
table.search(user_question).distance_threshold(0.5).limit(5)
            columns = ("id", "text", "_distance", "_score", "text_vec")
            st.dataframe(res.to_pandas(), column_order=columns,
hide_index=True)
            status.update(label="Retrieved relevant documents",
expanded=True)

        # Argument (A): Combine the retrieved chunks into the prompt.
        texts = [chunk.text for chunk in res.to_pydantic()]
        context = "\n\n".join(texts)
        prompt = PROMPT_TEMPLATE.format(question=user_question,
context=context)

        # Generation (G): Call the LLM to generate answer based on the
context.
        messages.append(
            {
                "role": "user",
                "content": prompt,
            }
        )
        llm_response = completion(
            api_base="http://localhost:11434",
            stream=True,
            model=LLM_MODEL,
            messages=messages,
        )
        final_response = st.write_stream(stream_text_only(llm_response))

    # Add assistant response to chat history
    st.session_state.messages.append({"role": "assistant", "content":
final_response})
```

Example 2 code:

In this demo, we will show you how to use hybrid search to combine vector search and full-text search on a set of documents.

app.py:

```python
import os

import dotenv
import streamlit as st

from pytidb import TiDBClient
from pytidb.schema import FullTextField, TableModel, Field
from pytidb.embeddings import EmbeddingFunction
from pytidb.rerankers import Reranker

# Load environment variables.
dotenv.load_dotenv()


# Connect to TiDB
db = TiDBClient.connect(
    host=os.getenv("TIDB_HOST", "localhost"),
    port=int(os.getenv("TIDB_PORT", "4000")),
    username=os.getenv("TIDB_USERNAME", "root"),
    password=os.getenv("TIDB_PASSWORD", ""),
    database=os.getenv("TIDB_DATABASE", "pytidb_hybrid_demo"),
    ensure_db=True,
)


# Create embedding function.
embed_fn = EmbeddingFunction(
    model_name="text-embedding-3-small",
    api_key=os.getenv("OPENAI_API_KEY"),
)


# Define reranker.
if os.getenv("JINA_AI_API_KEY") is not None:
    # Go to Jina AI Website (https://jina.ai/embeddings) to create a new
API key.
    reranker = Reranker(
        model_name="jina_ai/jina-reranker-v2-base-multilingual",
        api_key=os.getenv("JINA_AI_API_KEY"),
    )
else:
    reranker = None
```

```python
# Define table schema.
class Document(TableModel):
    __tablename__ = "documents"
    __table_args__ = {"extend_existing": True}

    id: int = Field(primary_key=True)
    text: str = FullTextField()
    text_vec: list[float] = embed_fn.VectorField(source_field="text")


# Sample documents
sample_documents = [
    "Ollama is an open-source platform that allows you to run large
language models (LLMs) locally on your machine.",
    "In the near future, a lonely writer develops an unlikely
relationship with an operating system designed to meet his every need.",
    "LangChain is a framework for developing AI applications powered by
large language models.",
    # For multi-language test.
    "TiDB is a database for AI applications with vector search,
knowledge graphs, and operational data capabilities.",
    "TiDB是一个开源的NewSQL数据库，支持混合事务和分析处理（HTAP）工作负载。",
    "TiDBはオープンソースの分散型HTAPデータベースで、トランザクション処理と分析処理
の両方をサポートしています。",
    "MySQL is a relational database management system.",
    "Redis is an in-memory data structure store.",
    "Docker containers package applications with their dependencies.",
    "HTTPS is a protocol for secure communication over the internet.",
    "SSH is a protocol for secure remote login from one computer to
another.",
    "Blockchain is a distributed ledger technology that enables secure
and transparent transactions.",
    "Autonomous vehicles are vehicles that can drive themselves without
human intervention.",
    "Linux is a family of open-source Unix-like operating systems based
on the Linux kernel.",
    "iOS is a smartphone operating system developed by Apple Inc.",
    "Android is an open-source mobile operating system developed by
Google.",
    "Twitter is a social media platform that allows users to share and
interact with messages called 'tweets'.",
    "Amazon is a platform for buying and selling products online.",
    "Azure is a cloud computing platform providing on-demand services to
individuals and organizations.",
]
```

```
table = db.open_table(Document)
if table is None:
    with st.spinner("Loading sample documents, it may take a while..."):
        table = db.create_table(schema=Document, if_exists="skip")
        if table.rows() == 0:
            table.bulk_insert([Document(text=text) for text in
sample_documents])

# Streamlit UI

# Sidebar.
with st.sidebar:
    st.logo(
        "../assets/logo-full.svg", size="large",
link="https://pingcap.github.io/ai/"
    )
    st.markdown(
        """#### Overview

**Hybrid search** fuses **exact matching** from full-text search with
**semantic understanding**
from vector search, delivering more relevant and reliable results.

    """
    )
    st.markdown("#### Settings")
    search_type = st.selectbox(
        label="Search type",
        options=["fulltext", "vector", "hybrid"],
        index=2,
    )
    distance_threshold = st.slider(
        label="Distance threshold",
        help="The vector distance between the query vector and the
vectors should be within this threshold.",
        min_value=0.0,
        max_value=1.0,
        value=0.7,
        disabled=search_type == "fulltext",
    )
    fusion_method_options = {
        "rrf": "Reciprocal Rank Fusion (RRF)",
        "weighted": "Weighted Score Fusion",
    }
    fusion_method = st.selectbox(
        label="Fusion method",
        options=fusion_method_options.keys(),
        format_func=lambda x: fusion_method_options[x],
        help="The method specifies how to fuse the scores from the
vector search and the full-text search.",
        index=0,
    )
    limit = st.slider(
        label="Limit",
        help="The number of documents to return.",
        min_value=1,
        max_value=20,
        value=10,
    )


# Main content.
st.markdown(
    '<h3 style="text-align: center; padding-top: 40px;">Hybrid Search
Demo</h3>',
    unsafe_allow_html=True,
)

with st.form("search_form"):
    col1, col2 = st.columns([6, 1])
    with col1:
        query_text = st.text_input(
            "Keywords:",
            placeholder="Enter your search query",
            label_visibility="collapsed",
        )
        st.markdown(
            'Try searching for: <b>"HTAP database"</b>',
            unsafe_allow_html=True,
        )
    with col2:
        submitted = st.form_submit_button("Search")

# Search results.
if submitted:
    with st.spinner("Searching documents containing the keyword..."):
        query = (
            table.search(query_text, search_type=search_type)
            .distance_threshold(distance_threshold)
            .fusion(method=fusion_method)
        )
        if reranker is not None:
            query = query.rerank(reranker, "text")

        df = query.limit(limit).to_pandas()

        if len(df) > 0:
            st.markdown(
                f'Found <b>{len(df)}</b> results for
<b>"{query_text}"</b>',
                unsafe_allow_html=True,
            )
            df = df.drop(columns=["text_vec"])
            st.dataframe(df, hide_index=True)
        else:
            st.empty()
```

example.py:

```
import json
import os
import dotenv

from pytidb import TiDBClient
from pytidb.embeddings import EmbeddingFunction
from pytidb.schema import FullTextField, TableModel, Field

dotenv.load_dotenv()


# Connect to TiDB.
print("=== CONNECT TO TIDB ===")
db = TiDBClient.connect(
    host=os.getenv("TIDB_HOST", "localhost"),
    port=int(os.getenv("TIDB_PORT", "4000")),
    username=os.getenv("TIDB_USERNAME", "root"),
    password=os.getenv("TIDB_PASSWORD", ""),
    database=os.getenv("TIDB_DATABASE", "hybrid_search_example"),
    ensure_db=True,
)
print("Connected to TiDB.\n")


# Create embedding function.
```

```
embed_fn = EmbeddingFunction(
    model_name="openai/text-embedding-3-small",
    api_key=os.getenv("OPENAI_API_KEY"),
)


# Create a table with a text field and a vector field.
print("=== CREATE TABLE ===")


class Chunk(TableModel, table=True):
    __tablename__ = "chunks"
    id: int = Field(primary_key=True)
    text: str = FullTextField()
    text_vec: list[float] = embed_fn.VectorField(source_field="text")


table = db.create_table(schema=Chunk, if_exists="overwrite")
print("Table created.\n")


# Insert some sample data.
print("=== INSERT SAMPLE DATA ===")
table.bulk_insert(
    [
        Chunk(
            text="TiDB is a distributed database that supports OLTP,
OLAP, HTAP and AI workloads.",
        ),
        Chunk(
            text="PyTiDB is a Python library for developers to connect
to TiDB.",
        ),
        Chunk(
            text="LlamaIndex is a Python library for building AI-powered
applications.",
        ),
    ]
)
print("Inserted 3 rows.\n")


# Perform hybrid search.
print("=== PERFORM HYBRID SEARCH ===")
results = table.search("AI database",
search_type="hybrid").limit(3).to_list()
```

```
print("Search results:")
for item in results:
    item.pop("text_vec")
print(json.dumps(results, indent=4, sort_keys=True))
```

Example 3 code:

Auto Embedding Demo
This example showcases how to use the auto embedding feature with PyTiDB Client.

- Connect to TiDB with PyTiDB Client
- Define a table with a VectorField configured for automatic embedding
- Insert plain text data, embeddings are populated automatically in the background
- Run vector searches with natural language queries, embedding happens transparently

main.py:

```
import os
import dotenv
from pytidb.embeddings import EmbeddingFunction
from pytidb.schema import TableModel, Field
from pytidb.datatype import TEXT
from pytidb import TiDBClient

# Load environment variables
dotenv.load_dotenv()

# Connect to database with connection parameters
tidb_client = TiDBClient.connect(
    host=os.getenv("TIDB_HOST", "localhost"),
    port=int(os.getenv("TIDB_PORT", "4000")),
    username=os.getenv("TIDB_USERNAME", "root"),
    password=os.getenv("TIDB_PASSWORD", ""),
    database=os.getenv("TIDB_DATABASE", "auto_embedding_example"),
    ensure_db=True,
)

# Define embedding function
print("=== Define embedding function ===")

provider = os.getenv("EMBEDDING_PROVIDER", "tidbcloud_free")
if provider == "tidbcloud_free":
    embed_func = EmbeddingFunction(
        model_name="tidbcloud_free/amazon/titan-embed-text-v2",
```

```python
    )
elif provider == "openai":
    tidb_client.configure_embedding_provider(
        provider="openai",
        api_key=os.getenv("OPENAI_API_KEY"),
    )
    embed_func = EmbeddingFunction(
        model_name="openai/text-embedding-3-small",
    )
elif provider == "jina_ai":
    tidb_client.configure_embedding_provider(
        provider="jina_ai",
        api_key=os.getenv("JINA_AI_API_KEY"),
    )
    embed_func = EmbeddingFunction(
        model_name="jina_ai/jina-embeddings-v3",
    )
elif provider == "cohere":
    tidb_client.configure_embedding_provider(
        provider="cohere",
        api_key=os.getenv("COHERE_API_KEY"),
        use_server=True,
    )
    embed_func = EmbeddingFunction(
        model_name="cohere/embed-v4.0",
        dimensions=1536,
        use_server=True,
    )
elif provider == "huggingface":
    tidb_client.configure_embedding_provider(
        provider="huggingface",
        api_key=os.getenv("HUGGINGFACE_API_KEY"),
    )
    embed_func = EmbeddingFunction(
        model_name="huggingface/sentence-transformers/all-MiniLM-L6-v2",
        dimensions=384,
        use_server=True,
    )
elif provider == "nvidia_nim":
    tidb_client.configure_embedding_provider(
        provider="nvidia_nim",
        api_key=os.getenv("NVIDIA_NIM_API_KEY"),
    )
    embed_func = EmbeddingFunction(
        model_name="nvidia_nim/baai/bge-m3",
        dimensions=1024,
        use_server=True,
    )
elif provider == "gemini":
    tidb_client.configure_embedding_provider(
        provider="gemini",
        api_key=os.getenv("GEMINI_API_KEY"),
    )
    embed_func = EmbeddingFunction(
        model_name="gemini/gemini-embedding-001",
        dimensions=3072,
        use_server=True,
    )
else:
    raise ValueError(f"Invalid embedding provider: {provider}")

print(f"Embedding function (model name: {embed_func.model_name})
defined")

# Define table schema
print("\n=== Define table schema ===")


class Chunk(TableModel):
    id: int = Field(primary_key=True)
    text: str = Field(sa_type=TEXT)
    text_vec: list[float] = embed_func.VectorField(source_field="text")


table = tidb_client.create_table(schema=Chunk, if_exists="overwrite")
print("Vector table created")

# Truncate table
print("\n=== Truncate table ===")
table.truncate()
print("Table truncated")

# Insert sample data
print("\n=== Insert sample data ===")
table.bulk_insert(
    [
        # 👇 The text will be embedded and the vector field will be
populated automatically.
        Chunk(
            text="TiDB is a distributed database that supports OLTP,
OLAP, HTAP and AI workloads."
        ),
        Chunk(text="PyTiDB is a Python library for developers to connect
to TiDB."),
        Chunk(
            text="LlamaIndex is a Python library for building AI-powered
applications."
        ),
    ]
)
print("Inserted 3 chunks")

# Perform vector search
print("\n=== Perform vector search ===")
results = (
    table.search("HTAP database").limit(3).to_list()
)  # 👈 The query text will be embedded automatically.
for item in results:
    print(f"id: {item['id']}, text: {item['text']}, distance:
{item['_distance']}")
```