vanna

Public

main 3 Branches 36 Tags Go to file t Go Add file Code

zainhoda Update README.md b2c8ecf · 2 days ago 354 Commits

.github/workflows	Delete .github/workflows/ci....	last month
docs	new notebooks	last month
img	readme updates	last month
nb-theme	new notebooks	last month
notebooks	update documentation for a...	4 days ago
papers	documentation updates	4 months ago
src/vanna	Update OpenAI class for Az...	4 days ago
tests	get related training data	5 months ago
training_data	add cybersyn-financial-data...	6 months ago
.gitignore	duckdb support	last week
.pre-commit-config.y...	base	5 months ago
CONTRIBUTING.md	update contributing md	last month
LICENSE	Initial commit	8 months ago
README.md	Update README.md	2 days ago
pyproject.toml	Update pyproject.toml	4 days ago
setup.cfg	remove unnecessary exclud...	5 months ago
tox.ini	Use tox in GH action + add [...	5 months ago

About

🤖 Chat with your SQL database 📊. Accurate Text-to-SQL Generation via LLMs using RAG 🔄.

vanna.ai/docs/

#agent #sql #database #ai #data-visualization #text-to-sql #rag #llm

Readme
MIT license
Activity
Custom properties
4.6k stars
30 watching
256 forks
Report repository

Releases 36

v0.0.36 Latest
4 days ago

+ 35 releases

Contributors 7

Deployments 124

github-pages 3 months ago

README MIT license

GitHub	PyPI	Documentation
GitHub vanna	pypi v0.0.36	Documentation vanna

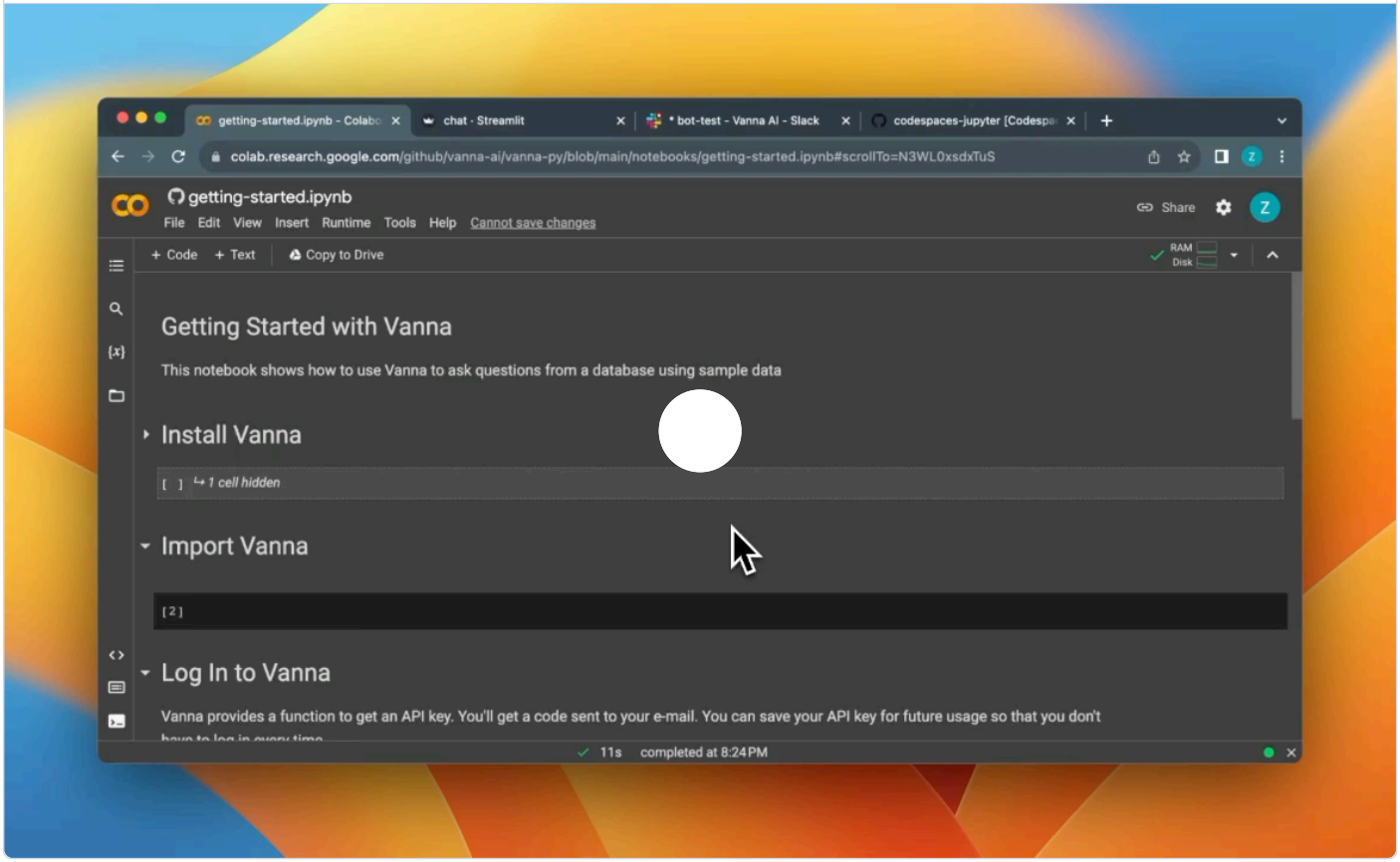
Languages

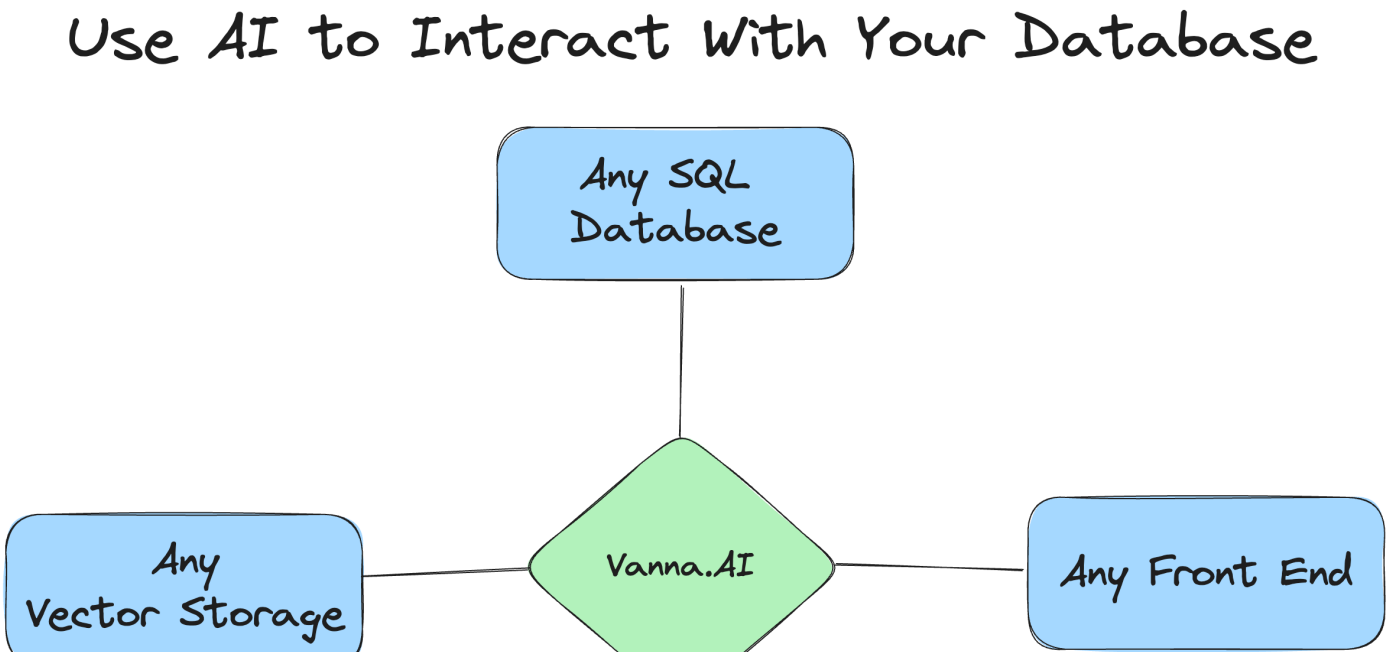
Jupyter Notebook 92.9%
Python 6.8%
Other 0.3%

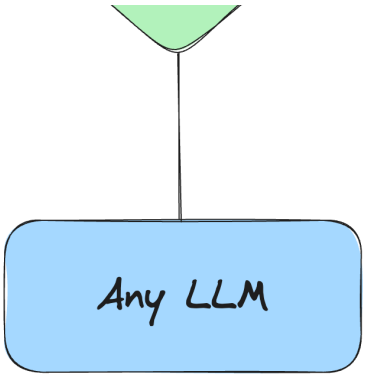
Vanna

Vanna is an MIT-licensed open-source Python RAG (Retrieval-Augmented Generation) framework for SQL generation and related functionality.

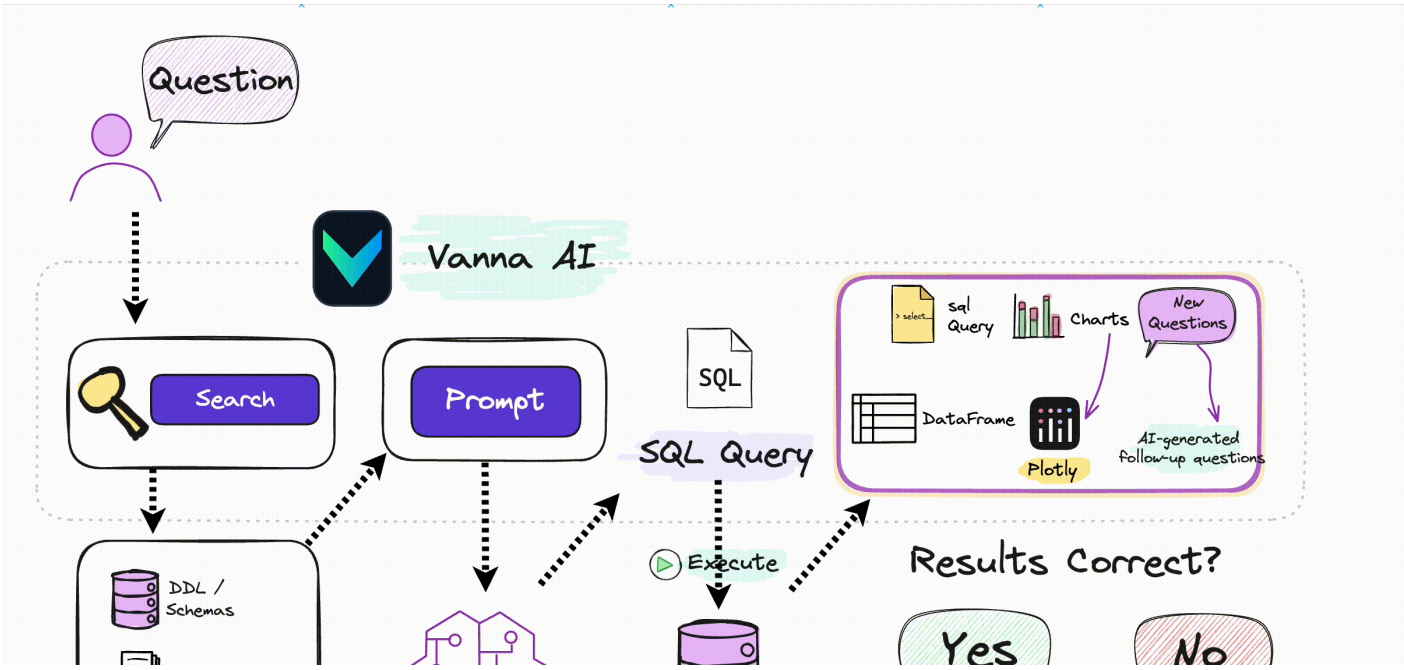
0802.mp4


getting-started.ipynb - Colabo
chat - Streamlit
* bot-test - Vanna AI - Slack
codespaces-jupyter [Codespa
colab.research.google.com/github/vanna-ai/vanna-py/blob/main/notebooks/getting-started.ipynb#scrollTo=N3WL0xsdxTuS
getting-started.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM
Disk
Getting Started with Vanna
This notebook shows how to use Vanna to ask questions from a database using sample data
Install Vanna
[ ] 1 cell hidden
Import Vanna
[2]
Log In to Vanna
Vanna provides a function to get an API key. You'll get a code sent to your e-mail. You can save your API key for future usage so that you don't
11s completed at 8:24PM


Use AI to Interact With Your Database
Any SQL Database
Any Vector Storage
Vanna.AI
Any Front End

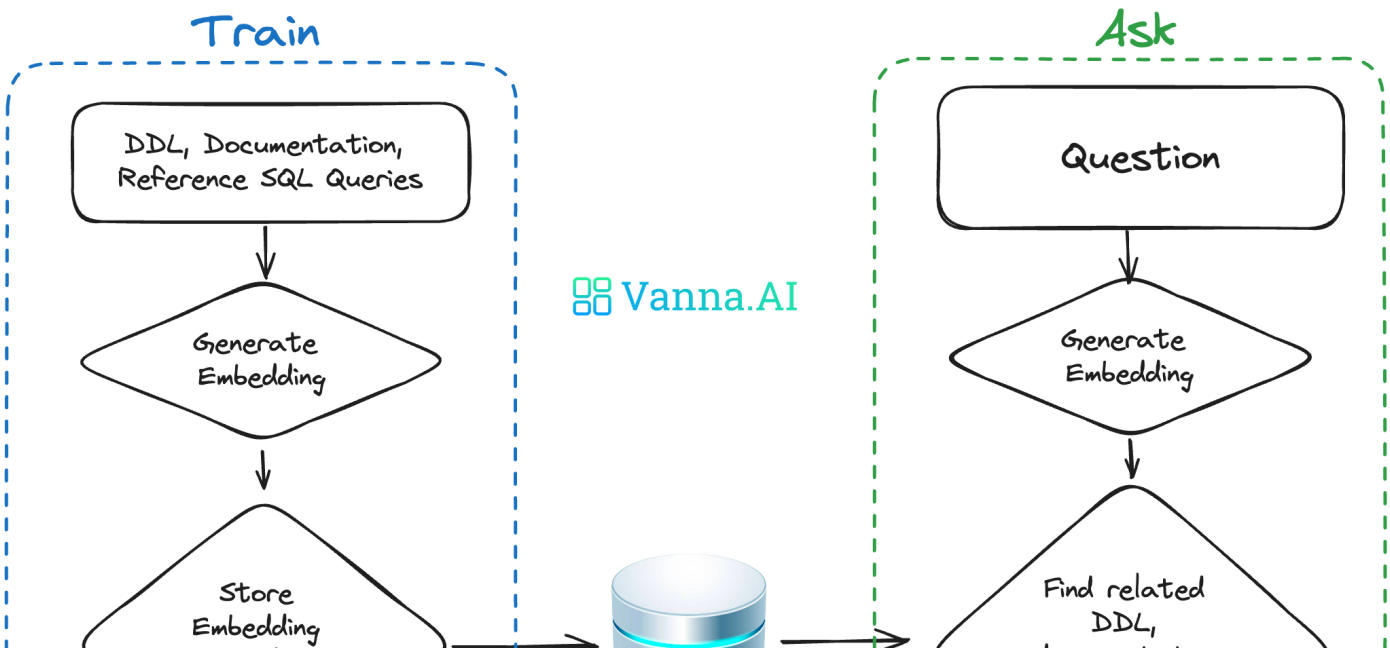


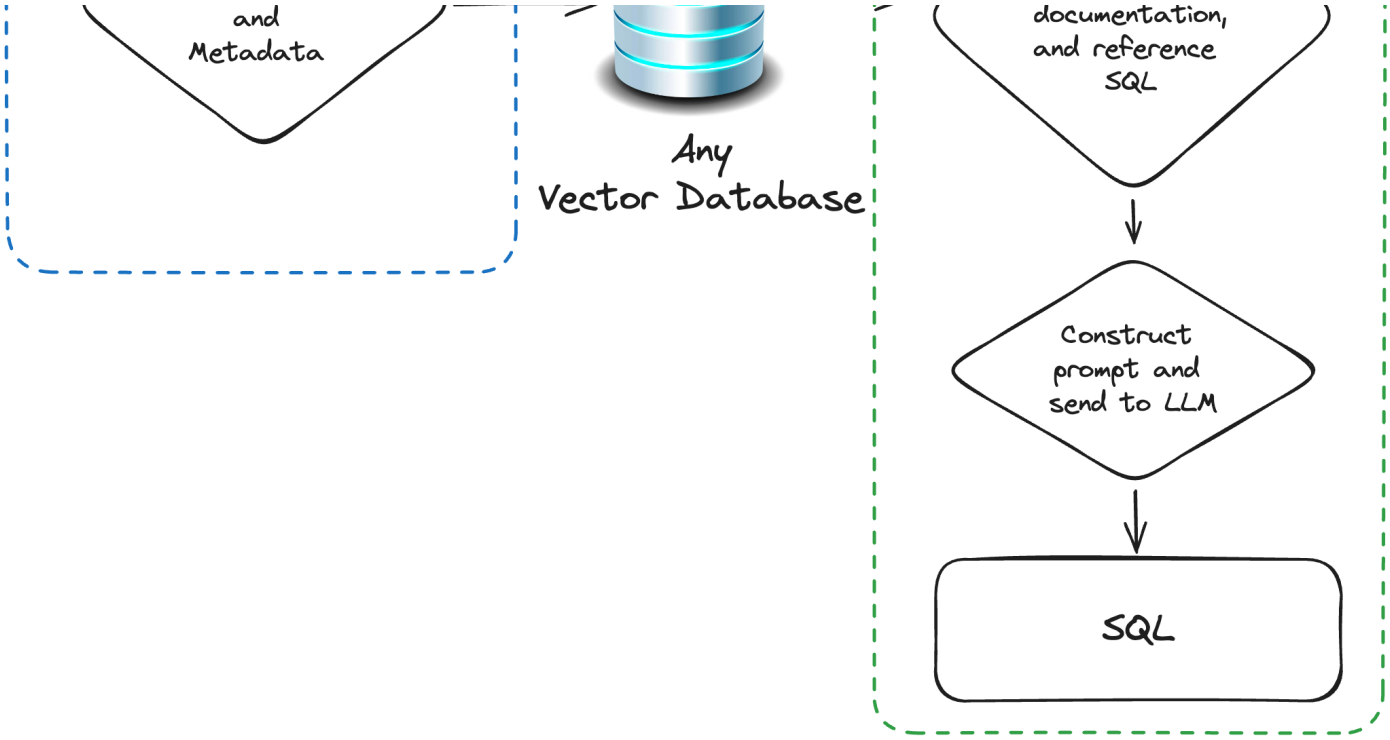
How Vanna works



Vanna works in two easy steps - train a RAG "model" on your data, and then ask questions which will return SQL queries that can be set up to automatically run on your database.

1. **Train a RAG "model" on your data**.
2. **Ask questions**.





If you don't know what RAG is, don't worry -- you don't need to know how this works under the hood to use it. You just need to know that you "train" a model, which stores some metadata and then use it to "ask" questions.

See the base class for more details on how this works under the hood.

User Interfaces

These are some of the user interfaces that we've built using Vanna. You can use these as-is or as a starting point for your own custom interface.

- Jupyter Notebook
- vanna-ai/vanna-streamlit
- vanna-ai/vanna-flask
- vanna-ai/vanna-slack

Getting started

See the documentation for specifics on your desired database, LLM, etc.

If you want to get a feel for how it works after training, you can try this Colab notebook.

Install

```
pip install vanna
```

There are a number of optional packages that can be installed so see the documentation for more details.

Import

See the documentation if you're customizing the LLM or vector database.

```
# The import statement will vary depending on your LLM and vector database. This is an example for OpenAI + Chrom
from vanna.openai.openai_chat import OpenAI_Chat
from vanna.chromadb.chromadb_vector import ChromaDB_VectorStore

class MyVanna(ChromaDB_VectorStore, OpenAI_Chat):
    def __init__(self, config=None):
        ChromaDB_VectorStore.__init__(self, config=config)
        OpenAI_Chat.__init__(self, config=config)

vn = MyVanna(config={'api_key': 'sk-...', 'model': 'gpt-4-...'})

# See the documentation for other options
```

Training

You may or may not need to run these `vn.train` commands depending on your use case. See the documentation for more details.

These statements are shown to give you a feel for how it works.

Train with DDL Statements

DDL statements contain information about the table names, columns, data types, and relationships in your database.

```
vn.train(ddl="""
    CREATE TABLE IF NOT EXISTS my-table (
        id INT PRIMARY KEY,
        name VARCHAR(100),
        age INT
    )
""")
```

Train with Documentation

Sometimes you may want to add documentation about your business terminology or definitions.

```
vn.train(documentation="Our business defines XYZ as ...")
```

Train with SQL

You can also add SQL queries to your training data. This is useful if you have some queries already laying around. You can just copy and paste those from your editor to begin generating new SQL.

```
vn.train(sql="SELECT name, age FROM my-table WHERE name = 'John Doe'")
```

Asking questions

```
vn.ask("What are the top 10 customers by sales?")
```

You'll get SQL

```
SELECT c.c_name as customer_name,
       sum(l.l_extendedprice * (1 - l.l_discount)) as total_sales
FROM   snowflake_sample_data.tpch_sf1.lineitem l join snowflake_sample_data.tpch_sf1.orders o
       ON l.l_orderkey = o.o_orderkey join snowflake_sample_data.tpch_sf1.customer c
       ON o.o_custkey = c.c_custkey
GROUP BY customer_name
```

```
ORDER BY total_sales desc limit 10;
```

If you've connected to a database, you'll get the table:

	CUSTOMER_NAME	TOTAL_SALES
0	Customer#000143500	6757566.0218
1	Customer#000095257	6294115.3340
2	Customer#000087115	6184649.5176
3	Customer#000131113	6080943.8305
4	Customer#000134380	6075141.9635
5	Customer#000103834	6059770.3232
6	Customer#000069682	6057779.0348
7	Customer#000102022	6039653.6335
8	Customer#000098587	6027021.5855
9	Customer#000064660	5905659.6159

You'll also get an automated Plotly chart:

RAG vs. Fine-Tuning

RAG

- Portable across LLMs
- Easy to remove training data if any of it becomes obsolete
- Much cheaper to run than fine-tuning
- More future-proof -- if a better LLM comes out, you can just swap it out

Fine-Tuning

- Good if you need to minimize tokens in the prompt
- Slow to get started
- Expensive to train and run (generally)

Why Vanna?

1. **High accuracy on complex datasets.**
    - Vanna's capabilities are tied to the training data you give it
    - More training data means better accuracy for large and complex datasets
2. **Secure and private.**
    - Your database contents are never sent to the LLM or the vector database
    - SQL execution happens in your local environment
3. **Self learning.**
    - If using via Jupyter, you can choose to "auto-train" it on the queries that were successfully executed
    - If using via other interfaces, you can have the interface prompt the user to provide feedback on the results
    - Correct question to SQL pairs are stored for future reference and make the future results more accurate
4. **Supports any SQL database.**
    - The package allows you to connect to any SQL database that you can otherwise connect to with Python
5. **Choose your front end.**
    - Most people start in a Jupyter Notebook.
    - Expose to your end users via Slackbot, web app, Streamlit app, or a custom front end.

Extending Vanna