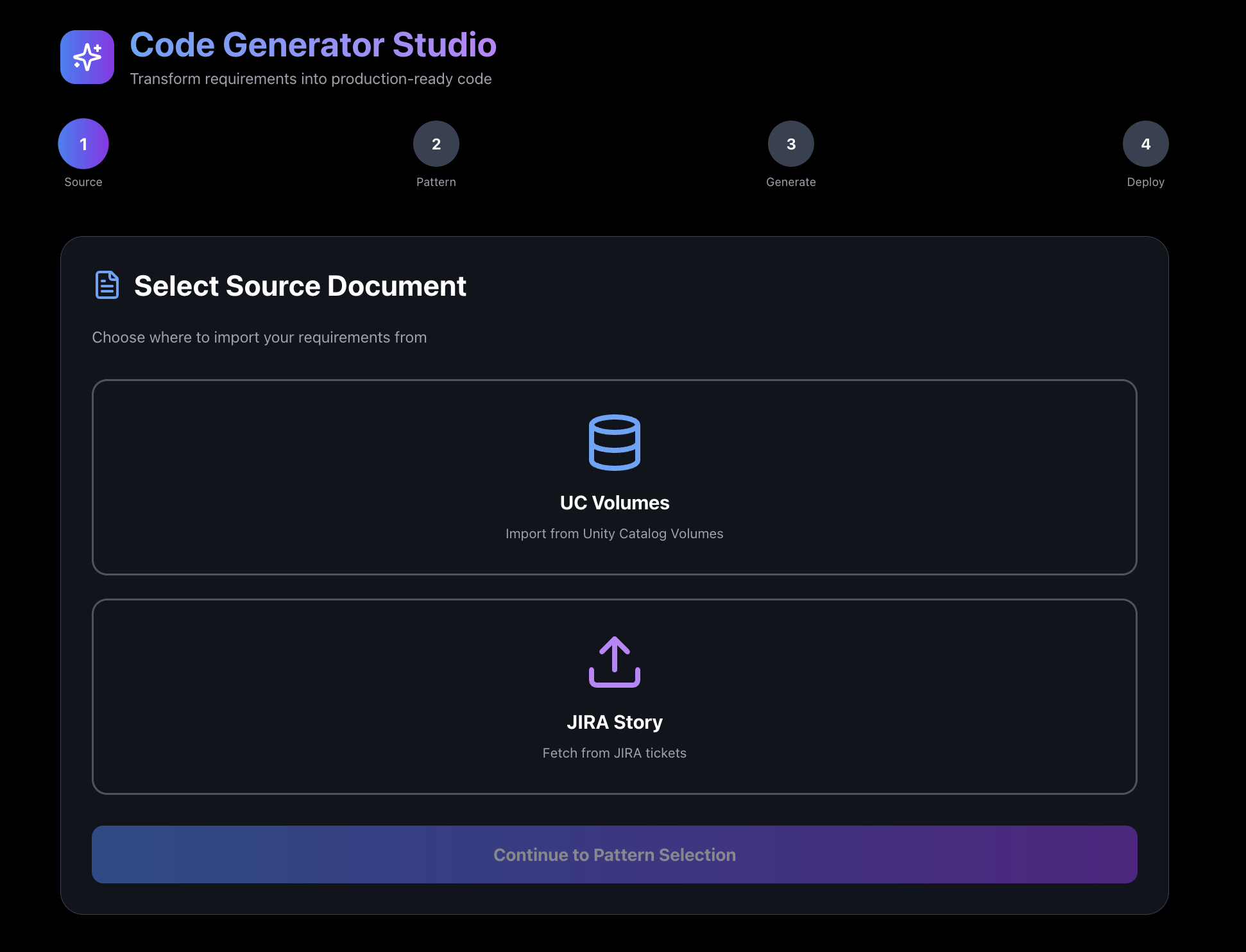
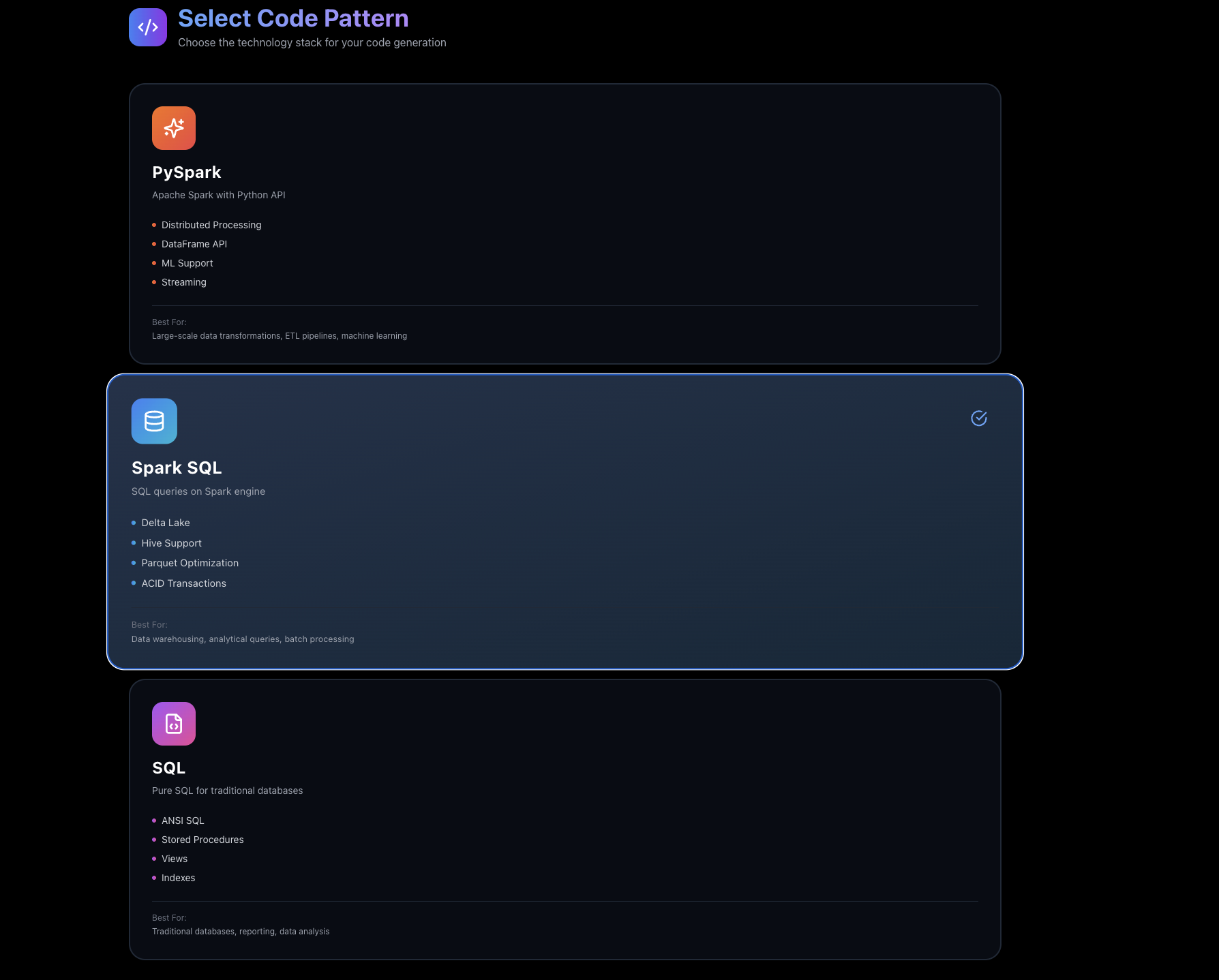
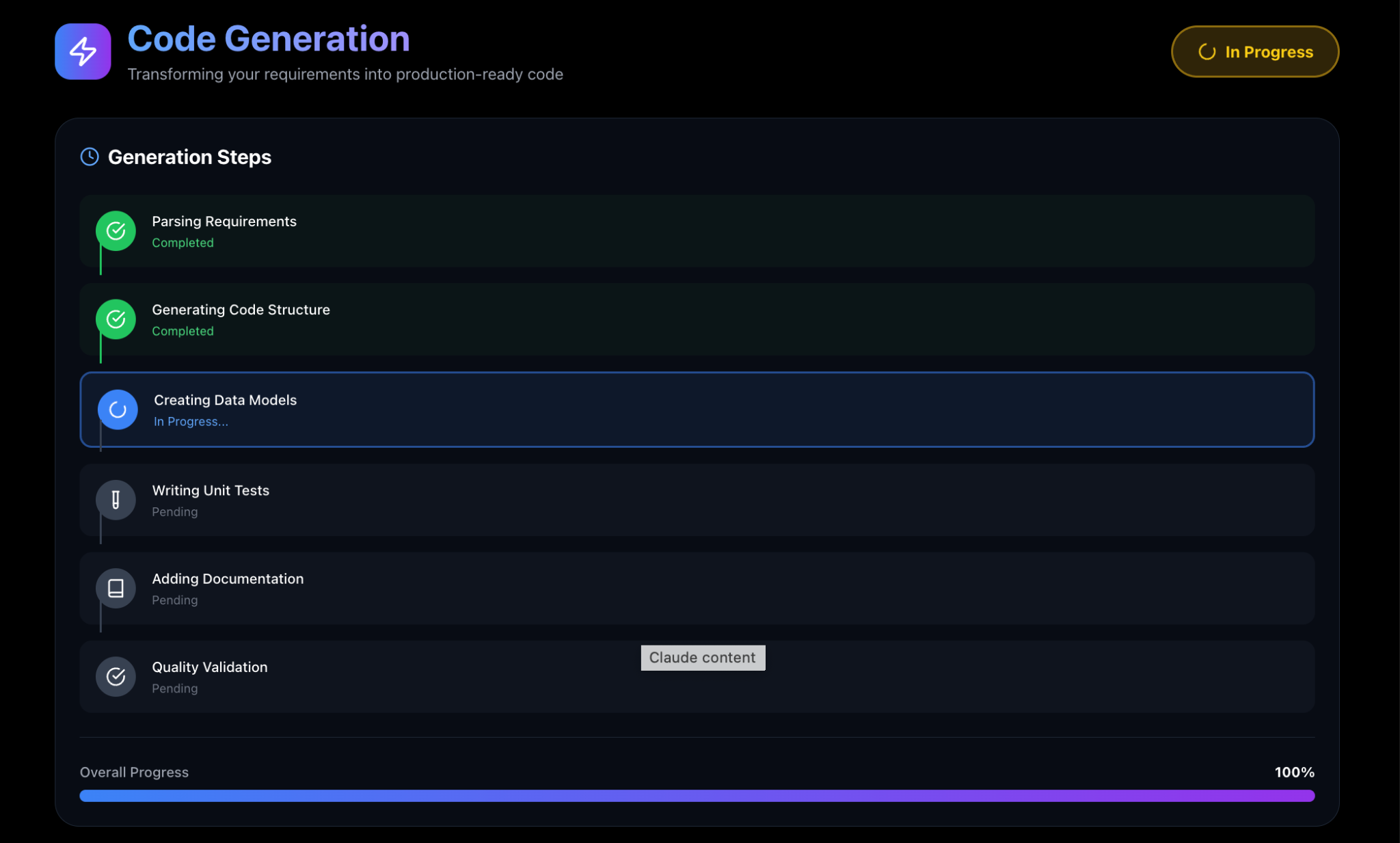
**Text to Code Generator Product Roadmap**

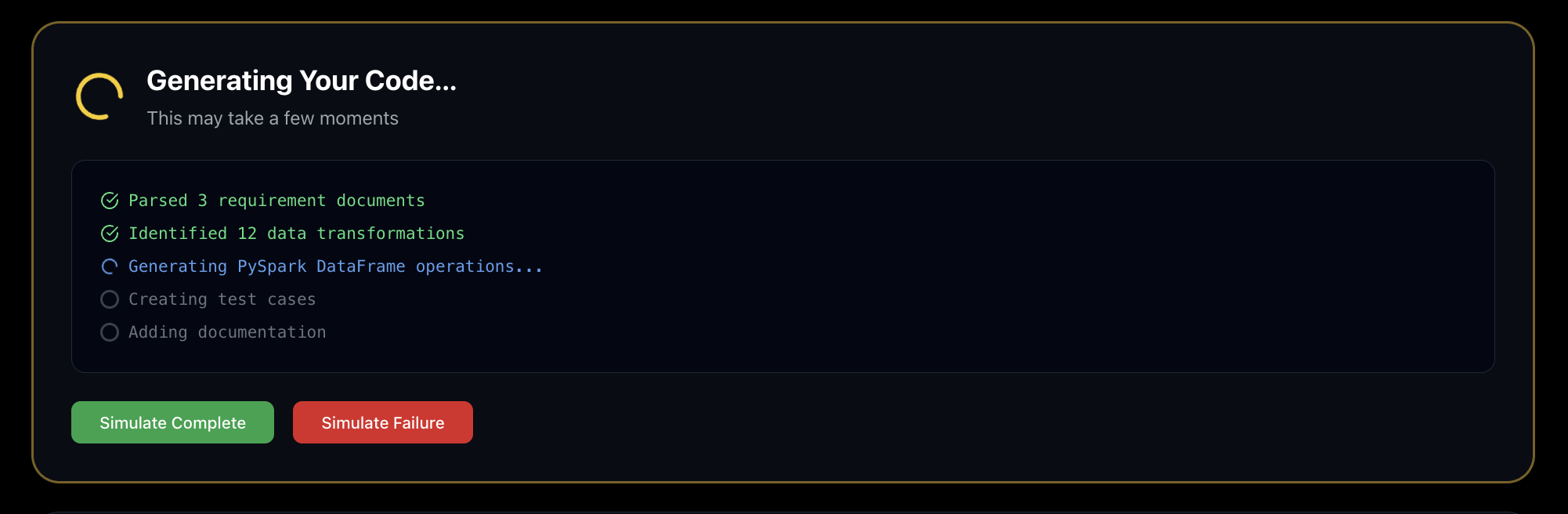
1. **Code Generation**

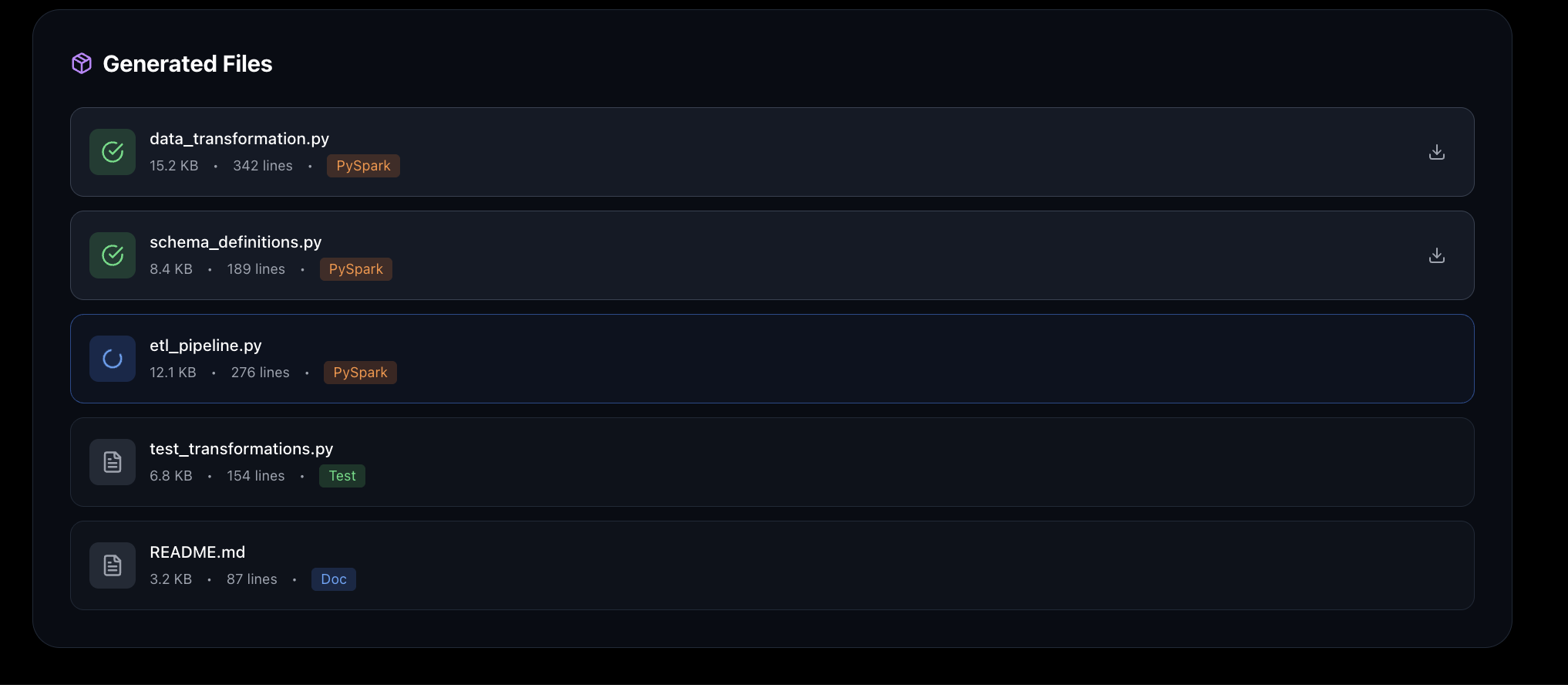


1. **Pattern Selection**

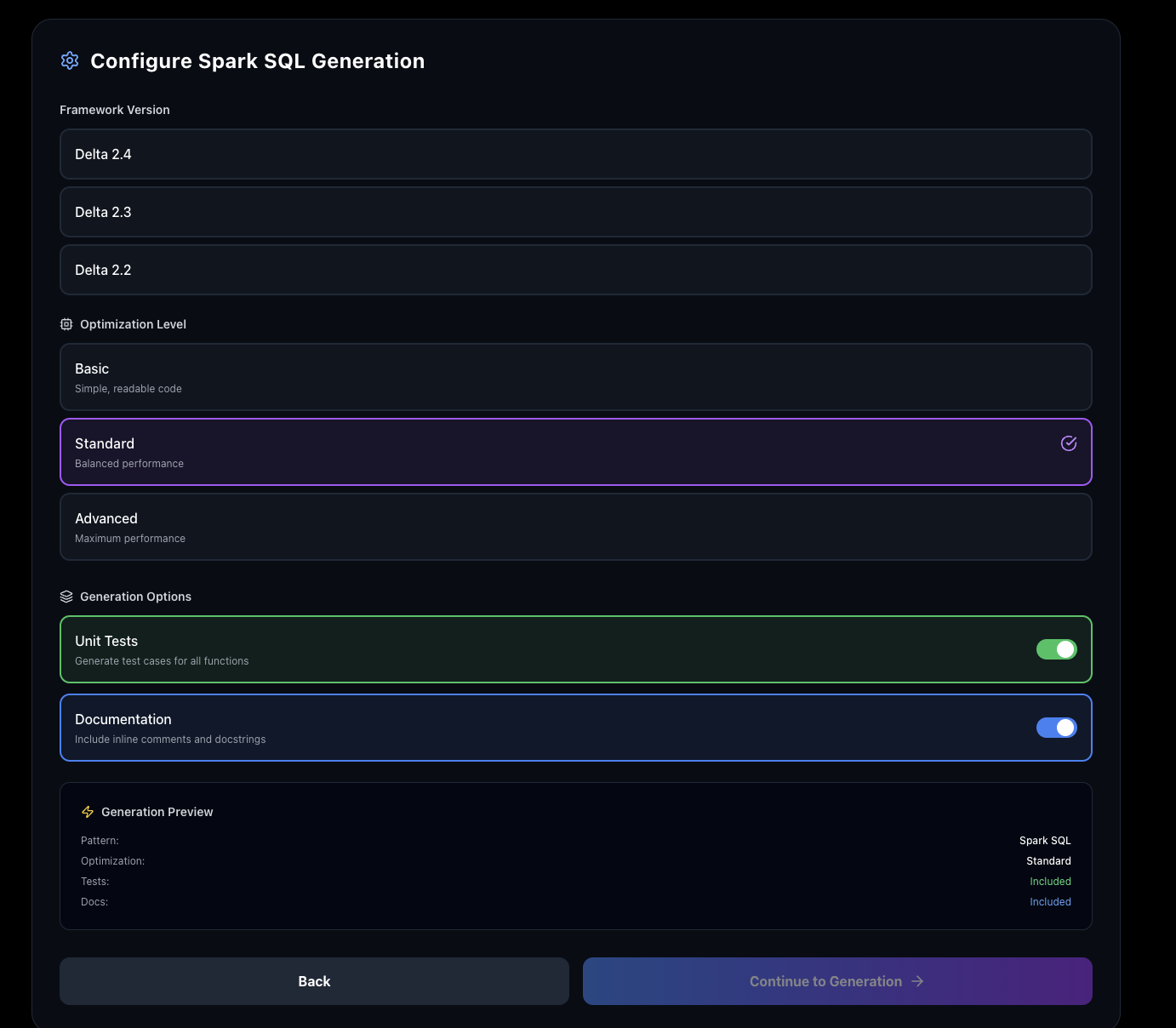




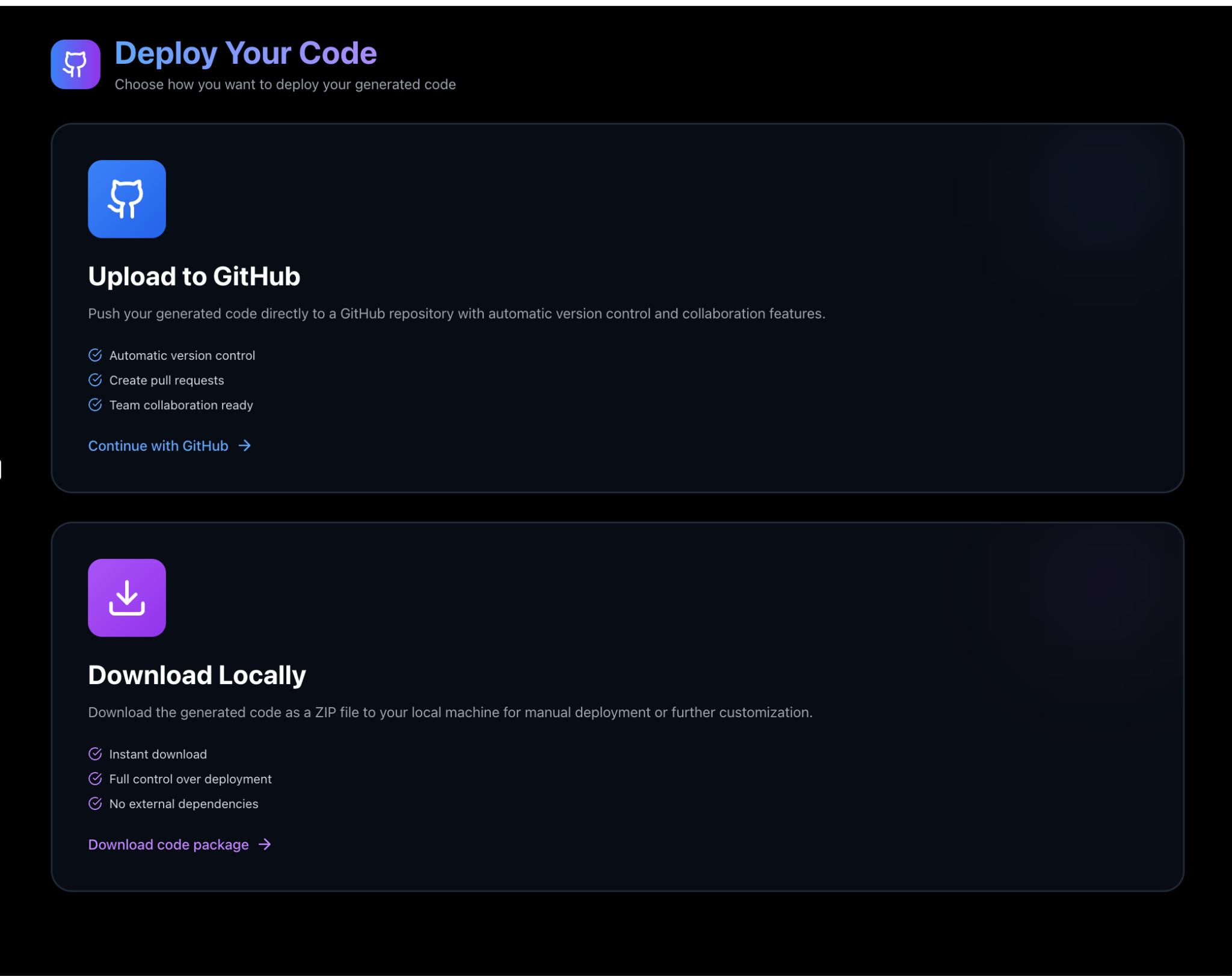




1. **Unit Test Case Generation**



**4. Deployment**



Code Snippet:  
  
# Databricks notebook: generate\_from\_volume\_mapping.py

# - Reads mapping CSV from UC Volume

# - Calls Databricks-hosted Claude Sonnet via OpenAI-style client using your PAT

# - Writes generated PySpark code to a file in a UC Volume

import os

import json

import textwrap

import traceback

from openai import OpenAI

from pyspark.sql import SparkSession

# ------------------ CONFIG ------------------

WORKSPACE\_URL = "https://adb-315278363237056.16.azuredatabricks.net" # <<-- adjust if different

MODEL\_NAME = "databricks-claude-sonnet-4-5" # <<-- your serving model name

MAPPING\_CSV\_PATH = "/Volumes/krish\_catalog/krish\_schema/test\_vol/input/mapping\_sheet.csv"

OUTPUT\_FILE\_PATH = "/Volumes/krish\_catalog/krish\_schema/test\_vol/output/output.py"

MIN\_RESPONSE\_LEN = 40 # sanity check for model output

# --------------------------------------------

spark = SparkSession.builder.getOrCreate()

# ---------- obtain Databricks PAT ----------

def get\_databricks\_token():

token = None

try:

# preferred: get token from running notebook context

token = dbutils.notebook.entry\_point.getDbutils().notebook().getContext().apiToken().get()

except Exception:

token = os.environ.get("DATABRICKS\_TOKEN")

if not token:

raise RuntimeError("Databricks token not found. Set DATABRICKS\_TOKEN env var or run inside a Databricks notebook.")

return token

# ---------- read mapping CSV from the volume ----------

def read\_mapping\_csv(path):

"""

Expects CSV with header:

Source\_Table,Source\_Column,Target\_Table,Target\_Column,Transformation

Returns: list of mappings grouped by Source\_Table -> list of column mappings

"""

print(f"Reading mapping CSV from: {path}")

sdf = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load(path)

pdf = sdf.toPandas()

# group by Source\_Table to produce one mapping per source table

mappings = {}

for \_, row in pdf.iterrows():

src\_tbl = str(row.get("Source\_Table") or "").strip()

src\_col = str(row.get("Source\_Column") or "").strip()

tgt\_tbl = str(row.get("Target\_Table") or "").strip()

tgt\_col = str(row.get("Target\_Column") or "").strip()

transform = str(row.get("Transformation") or "").strip() # optional expression

if not src\_tbl:

continue

mappings.setdefault(src\_tbl, []).append({

"source\_column": src\_col,

"target\_table": tgt\_tbl,

"target\_column": tgt\_col,

"transformation": transform

})

# convert to list of mapping objects (one per source table)

mapping\_list = []

for src\_tbl, cols in mappings.items():

# prefer target\_table from the first row (assumes same target for all rows of same source)

mapping\_list.append({

"pipeline\_id": src\_tbl.replace(".", "\_"),

"source\_path\_or\_table": src\_tbl, # the generated code will read from this path/table (we'll instruct model to treat as volume path)

"columns": cols

})

return mapping\_list

# ---------- build prompt for a single mapping ----------

def build\_prompt\_for\_mapping(map\_obj):

"""

map\_obj example:

{

"pipeline\_id": "...",

"source\_path\_or\_table": "/Volumes/krish\_catalog/.../input/…",

"columns": [

{"source\_column":"a","target\_table":"/Volumes/.../silver...","target\_column":"A","transformation":"a\*2"},

...

]

}

"""

instr = textwrap.dedent("""

You are an expert Databricks PySpark engineer. Generate exactly one Python PySpark script (only raw python code, no markdown)

that:

- Reads data from the SOURCE path (this is a Unity Catalog Volume path) using spark.read.format(...).load(source\_path)

- Applies per-column transformations (provided below). For each mapping row:

- If transformation is non-empty, compute target\_column as the transformation expression using DataFrame APIs (withColumn/selectExpr)

- Otherwise map source\_column -> target\_column directly

- Writes final result as Delta into the TARGET VOLUME path (use Delta write to the target path)

- If more than one unique target\_table exists in the mapping rows, assume they are all the same volume path; if not, write to the first target path

- If Transformation describes a derived column with no Source\_Column (empty Source\_Table/Source\_Column), create the column based on the target column name and load data based on the transformation logic.

- Parameterize run\_date and max\_ts with dbutils.widgets if required (but not mandatory)

- Add minimal error handling and logging

- Use only pyspark + delta APIs (no external SDKs)

- Use DeltaTable.forPath only if implementing MERGE. If a merge is not requested, use write.format('delta').mode('overwrite').save(target\_path)

- Keep code deterministic and idiomatic (clear variable names)

- Output a single file content (the generated script)

""").strip()

# Provide mapping rows in JSON for model to use

payload = {

"pipeline\_id": map\_obj.get("pipeline\_id"),

"source\_path\_or\_table": map\_obj.get("source\_path\_or\_table"),

"mappings": map\_obj.get("columns")

}

prompt = instr + "\n\nMapping JSON:\n" + json.dumps(payload, indent=2) + "\n\nNow output the Python script only."

return prompt

# ---------- call Claude Sonnet via OpenAI-style client ----------

def call\_claude\_sonnet(prompt, token):

client = OpenAI(api\_key=token, base\_url=f"{WORKSPACE\_URL}/serving-endpoints")

# Compose messages: system + user

messages = [

{"role": "system", "content": "You are a senior Databricks PySpark engineer."},

{"role": "user", "content": prompt}

]

resp = client.chat.completions.create(

model=MODEL\_NAME,

messages=messages,

max\_tokens=4000,

temperature=0.05

)

# robustly extract text

try:

return resp.choices[0].message.content

except Exception:

# fallback stringify

return str(resp)

# ---------- save generated code to the target volume path (multiple attempts) ----------

def save\_code\_to\_volume(target\_path, code\_text, pipeline\_id="unnamed"):

"""

Attempt several common methods to write a single file into a Volume path.

1) Try dbutils.fs.put(target\_path, content, overwrite=True)

2) Try writing via '/dbfs' prefix (local file write)

3) Fallback: write to /dbfs/FileStore and attempt dbutils.fs.cp to move it to target path

Note: volume paths behavior depends on your workspace config; this tries common approaches and prints helpful messages.

"""

print(f"Attempting to save generated code to: {target\_path}")

# attempt 1: dbutils.fs.put (works for many path schemes)

try:

dbutils.fs.put(target\_path, code\_text, True)

print(f"Saved via dbutils.fs.put -> {target\_path}")

return True

except Exception as e:

print("dbutils.fs.put failed:", e)

# attempt 2: write to local /dbfs path if available

try:

if target\_path.startswith("/"):

dbfs\_local = "/dbfs" + target\_path # /dbfs + /Volumes/...

else:

dbfs\_local = "/dbfs/" + target\_path

os.makedirs(os.path.dirname(dbfs\_local), exist\_ok=True)

with open(dbfs\_local, "w", encoding="utf-8") as f:

f.write(code\_text)

print(f"Saved via local /dbfs path -> {dbfs\_local}")

return True

except Exception as e:

print("/dbfs write failed:", e)

# attempt 3: write to FileStore and try to cp to target via dbutils

try:

tmp\_local = f"/dbfs/FileStore/generated\_pipelines/{pipeline\_id}.py"

os.makedirs(os.path.dirname(tmp\_local), exist\_ok=True)

with open(tmp\_local, "w", encoding="utf-8") as f:

f.write(code\_text)

# try to copy from file:/ to the target path using dbutils

src\_uri = "file:" + tmp\_local

try:

dbutils.fs.cp(src\_uri, target\_path)

print(f"Saved via FileStore + dbutils.fs.cp -> {target\_path}")

return True

except Exception as e\_cp:

print("dbutils.fs.cp failed:", e\_cp)

print(f"Generated file is available at {tmp\_local} (FileStore). Please move it manually to the volume.")

return False

except Exception as e:

print("Fallback FileStore write failed:", e)

return False

# ------------------ main ------------------

def main():

try:

token = get\_databricks\_token()

mapping\_list = read\_mapping\_csv(MAPPING\_CSV\_PATH)

if not mapping\_list:

print("No mapping rows found. Exiting.")

return

# For this request you said: write the pyspark generated code in another volume as one file output.py

# If mapping\_list has multiple source tables, we will combine them into a single generated script file.

# If you prefer one file per pipeline, adapt logic to loop and create one file each.

# Combine mappings into a single 'document' for the model

combined = {

"pipelines": mapping\_list,

"note": "Generate a single Python file that iterates over each pipeline mapping and runs the ETL for each mapping. The file should be parameterized and idempotent."

}

# Build one prompt for entire mapping set (single output file)

# Alternatively you could iterate mapping\_list and generate separate files per pipeline.

prompt = textwrap.dedent("""

You are an expert Databricks PySpark engineer. Generate a single Python PySpark script (output the raw python code only)

that:

- Iterates over each pipeline mapping provided in the JSON below.

- For each mapping: read the source (a Unity Catalog Volume path) using spark.read.format(...).load(source\_path),

apply per-column transformations (or pass-through), and write the results to the target Unity Catalog Volume path using Delta.

- Use clear function boundaries: read\_source, apply\_transformations, write\_target.

- Add basic logging and error handling but avoid secrets.

- The generated script must be runnable in Databricks (use dbutils.widgets for parameters if needed).

""").strip() + "\n\nMappings:\n" + json.dumps(combined, indent=2) + "\n\nNow output the Python script only."

# Call model

model\_response = call\_claude\_sonnet(prompt, token)

code\_text = model\_response if isinstance(model\_response, str) else str(model\_response)

# sanity check

if not code\_text or len(code\_text.strip()) < MIN\_RESPONSE\_LEN:

raise RuntimeError("Model returned empty or too-short response; aborting.")

# save to the requested output volume file path

saved = save\_code\_to\_volume(OUTPUT\_FILE\_PATH, code\_text, pipeline\_id="all\_pipelines")

if saved:

print(f"✅ Successfully saved generated code to {OUTPUT\_FILE\_PATH}")

else:

print("⚠️ Could not save directly to volume. Generated file is available under /dbfs/FileStore/generated\_pipelines/ if fallback wrote it there.")

except Exception as e:

print("Fatal error:", e)

traceback.print\_exc()

# run

if \_\_name\_\_ == "\_\_main\_\_":

main()