



devoteam
Snowflake Partner

CONSULTANT SELF-TRAINING

TRAINING DECK · STREAMLIT-COCO · V0.1.5

streamlit-coco

Consultant **Self-Training.**

Go through this deck alone, do the hands-on lab, and emerge ready to demo, build, and deliver `streamlit-coco` on a client engagement. Companion files: `doc/training/*.md`.

What is **streamlit-coco**?

One sentence: the elevator pitch

A Python library that wraps Snowflake's **Cortex Code Agent SDK ("CoCo")** into production-ready Streamlit components — streaming agent UI, readable tool cards, and human-in-the-loop approval gates, in about 10 lines of code.

- **PyPI package**

```
pip install streamlit-coco[sdk]
```

- **License**

Apache-2.0, open source

The problem it solves

Why this library exists

Teams building Streamlit apps on Snowflake want AI-assisted data exploration, code generation, and automated workflows. Wiring Cortex Code into a custom Streamlit app from scratch means building session management, streaming rendering, tool-card display, and safety gates yourself — weeks of plumbing with no reusable foundation.

Without streamlit-coco	With streamlit-coco
Wire CoCo yourself across Streamlit reruns	Session + fragment polling that keeps streaming
Raw JSON tool dumps	Meaningful cards (Glob, Grep, Read, Write, SQL, AskUser...)
Hope the agent behaves	<code>require_approval_for</code> + HITL UI
Chat-only demos	Structured callbacks into your own widgets

How it fits the **Snowflake ecosystem**

Who built it, current maturity

Position

- Cortex Code today is consumed via CLI, Showsight, Desktop, IDE extensions
- streamlit-coco bridges the gap into custom Streamlit apps
- Pure Python — no JS build step, works with SiS-style local Streamlit

Maturity & ownership

- **Level**
N0 (alpha), targeting N1 — Snow Builders
- **Owner**
Laurent Letourmy — Devoteam Snowflake Partner
- **Version**
0.1.5 on PyPI

Current adoption & **outcomes**

Snapshot as of the roadmap's last success-check

107

PyPI downloads last month

2

GitHub stars

0.1.5

Current alpha version

Honest state (be transparent with clients)

This is alpha software, 0 client engagements to date, and targets are still ahead of current usage (500+ downloads/month, 50+ stars, 3+ community examples). Position it accordingly: strong internal-tooling fit today, production readiness targeted at N1. Do not oversell maturity in a client conversation.

Typical **engagement modes**

Three ways teams plug this in

INTERACTIVE

Analyst copilot

`panel()` + `chat_input_bar()` embedded in a dashboard; analysts ask questions, agent queries Snowflake with approval gates.

STRUCTURED

Self-updating widgets

Agent output routed into your own dataframes/charts via `on_structured_output` — no chat window needed.

HEADLESS

CI / pipeline step

`query()` in a script — no Streamlit import at all; embed CoCo in scheduled jobs or CI.

What you'll be able to **do**

Concrete outcomes after this deck + lab

- **Explain it**

What it does, when to use it vs. a bare CoCo CLI/Snowsight session

- **Run every demo**

make chat, make backlog, make headless, make structured, make cwd-upload

- **Build from scratch**

A minimal app using panel() + chat_input_bar(), approval gates configured

- **Present a demo**

Deliver the 12-minute scripted demo to a client or internal audience unassisted

- **Map to a project**

Presales demo, delivery sprint (Assess→Build→Enable), or client workshop (W01)

How it **works**

"You own the page. CoCo owns the session."

Flow (interactive mode)

`check_environment()` → verify SDK / CLI / Snowflake connection are ready, without starting an agent
`render_start_gate()` → shows status, only starts the session after you click "Start"
`get_or_create_session()` → creates/retrieves a `CocoSession` tied to a Streamlit session key
`panel()` + `chat_input_bar()` → renders the streaming transcript and your input, survives Streamlit reruns

● Under the hood: the SDK spawns the CoCo CLI as a subprocess and streams NDJSON events

● Same event model powers `panel()`, structured output, and `headless_query()`

Key components

The public API surface you'll actually use

Capability	Entry points
Native panel + approvals	<code>panel()</code> , <code>chat_input_bar()</code> , <code>render_approvals()</code>
Tool cards & AskUser / plan UI	<code>doc/features/tools-display/</code>
Session & options	<code>CocoSession</code> , <code>CocoOptions</code> , <code>get_or_create_session</code>
Headless events	<code>query()</code> , <code>CocoSession.stream()</code> , <code>execute_plan()</code>
File upload into cwd (0.1.5)	<code>upload_to_cwd()</code> , <code>cwd_uploader()</code>
Legacy CCv2	<code>chat()</code>

Prerequisites

Full detail: [doc/training/setup-guide.md](#)

- Python **3.10+** and `uv` installed
- A Snowflake account with **Cortex Code** access
- A working `~/ .snowflake/connections.toml` (or CLI default connection)
- The **CoCo CLI** (`cortex`) installed and on `PATH`
- Git access to `DevoteamSP/streamlit-coco-dev`

Install & connect

Copy-paste ready

```
git clone https://github.com/DevoteamSP/streamlit-coco-dev.git
cd streamlit-coco-dev
make install      # uv sync --extra dev
cortex --version # verify CoCo CLI
```

Then set up `~/.snowflake/connections.toml`:

```
[connections.analytics]
account = "xyl2345"
user = "you@company.com"
authenticator = "externalbrowser"
```

Verification checklist

You're ready for the lab when...

- `import streamlit_coco as c; print(c.__version__)` prints `0.1.5`

- `cortex --version` succeeds

- `check_environment(...).ready` is `True`

- `make check` passes with no failures

- `make chat` opens a working app with a streamed response + tool card

First impression — **panel()**

Full exercise: hands-on-lab.md Exercise 1

```
make chat # runs examples/chat_app.py
```

● Type: "What tables are available in SNOWFLAKE_SAMPLE_DATA.TPCH_SF1?"

● Watch the transcript stream progressively, not all at once

● **Check:** at least one tool card appears showing what the agent did

Approval **gates**

Full exercise: hands-on-lab.md Exercise 2

- Type: "Show me the top 5 customers by total order amount. Write the SQL and run it."
- Approval banner fires — read the full SQL, click **Approve once**
- Type: "Now create a summary table called TOP_CUSTOMERS." → click **Deny** with a reason

This is the core differentiator: every destructive tool (SQL, Write, Edit, Bash) pauses for an explicit human decision. No surprises.

Structured **output**

Full exercise: hands-on-lab.md Exercise 3

```
make structured # runs examples/structured_output.py

def render(data: dict, result: st_coco.CocoChatResult) -> None:
    st.dataframe(data.get("selected_features", []))

st_coco.panel(session=session, on_structured_output=render)
```

- Type: "Give me a breakdown of orders by status" → renders as a chart, not raw JSON

Headless **mode**

Full exercise: hands-on-lab.md Exercise 4

```
import asyncio
import streamlit_coco as coco

async def run():
    async for event in coco.query("Profile ANALYTICS.CUSTOMERS"):
        if event.type == "result":
            print(event.structured_output)

asyncio.run(run())  # or: make headless
```

- No `streamlit` import anywhere — same `query()`, same event types, works in CI/scripts

Build from **scratch**

Full exercise: hands-on-lab.md Exercise 5 — the real test

Using only the README quickstart (no copy-paste from examples/), write:

```
opts = st_coco.CocoOptions(  
    connection="analytics", cwd=".",  
    allowed_tools=["Read", "Glob", "Grep"],  
    require_approval_for=["Edit", "Write", "Bash"],  
)  
env = st_coco.check_environment(connection=opts.connection)  
if not st_coco.render_start_gate(opts, session_key="copilot", env=env):  
    st.stop()  
session = st_coco.get_or_create_session(opts, key="copilot")  
st_coco.panel(session=session, warm_up=True, show_status=True)  
st_coco.chat_input_bar(session, placeholder="Ask CoCo...")
```

Before you **demo**

Full script: `doc/training/demo-script.md`

- Snowflake account with Cortex Code access, connection configured
- `cortex --version` succeeds; `make check` passes
- A database with sample tables (e.g. SNOWFLAKE_SAMPLE_DATA.TPCH_SF1)
- Terminal + browser side by side; **dry-run once within 24h of the real demo**

Opening hook **(30s)** + Act 1

~2 min

"What if your Streamlit app had a built-in AI coding agent that can query Snowflake, read files, and write code — with guardrails? streamlit-coco gives you that in about 10 lines of code."

- Show `examples/chat_app.py` (~20 lines) → run `make chat` → ask about TPCH_SF1 tables

Act 2 — Approval gates

~3 min

● "Show me the top 5 customers by total order amount. Write the SQL and run it." → Approve

● "Now create a summary table called TOP_CUSTOMERS." → Deny with reason

Talking point: "Every destructive action — SQL, file writes, bash commands — requires explicit human approval. No surprises."

Act 3+4 — **Output & headless**

~2 min (headless optional if time is tight)

- `make structured` → "Give me a breakdown of orders by status" → renders as a chart

- `make headless` → terminal streams events, no browser

Talking point: "Agent output flows into your widgets. CI pipelines, scheduled jobs — embed CoCo anywhere Python runs."

Closing & CTA

~2.5 min

- `pip install streamlit-coco[sdk]` — 5 minutes to first working app

- Safety by default — approval gates on for all destructive tools

- Flexible — interactive panel, structured output, or headless, same library

CTA: "Try it against your own Snowflake account this week — the README quickstart gets you to a working app in under 10 minutes."

Top mistakes **new users make**

Save yourself the debugging time

- Listing the same tool in both `allowed_tools` and `require_approval_for` — don't overlap them
- Assuming unlisted tools auto-run — the default is **require approval** unless explicitly allowed
- Expecting Streamlit Community Cloud or SiS support today — CoCo needs a subprocess + CLI on the same host
- Skipping `check_environment()` and debugging a stuck session blind — always check readiness first
- Positioning this as production-ready to a client — it's alpha (0.1.5), be upfront

FAQ

Answers you'll actually be asked in a demo

Does it work with SiS?	Not yet — pure Python, waiting on the CoCo API path; see roadmap.
What LLM does it use?	Whatever Cortex Code uses under the hood — no config needed.
Can I customize approvals?	Yes — <code>allowed_tools</code> / <code>require_approval_for</code> in <code>CocoOptions</code> .
Is it production-ready?	Alpha (0.1.5) — fine for internal tools; production readiness targeted at N1.
Vs. the CoCo CLI directly?	This library owns the Streamlit-specific plumbing (session, streaming, cards, approvals) — the CLI alone has none of that.

5 things to **remember**

Core principles

01 "You own the page, CoCo owns the session" — `panel()/chat_input_bar()` is the preferred pattern

02 Safety by default: unlisted tools require approval; be deliberate about `allowed_tools`

03 Three modes, one library: interactive panel, structured output, `headless query()`

04 This is alpha (0.1.5) — be honest about maturity with clients, no client engagements yet

05 Golden-path checklists live in `doc/features/` — run them before signing off a release demo

Where to go **next**

Companion files for hands-on practice

Training pack (doc/training/)

`setup-guide.md`

`hands-on-lab.md`

`demo-script.md`

`quiz.md`

Repo docs

`README.md` (quickstart)

`doc/deployment/local.md`

`doc/api.md` · `doc/roadmap.md`

`doc/marketing/demo.md` (source of truth for the client demo)



devoteam
Snowflake Partner

Take the **quiz** next.

Open `doc/training/quiz.md` — score 8/10 or higher and you're considered trained for this asset's N0→N1 gate item.

Questions? Laurent Letourmy — asset owner