DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Module 3

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Deploying Your Prototype

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

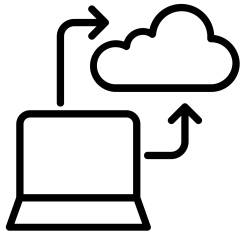
Choosing the Right Deployment Strategy

MVP Build Plan

1. Getting data into Snowflake
2. Cleanup Data
3. Analyze Data
4. Visualize Results
5. Create interface with a tab called "Data" that displays the head of the dataset
6. Deploy to cloud

Two Main Deployment Options

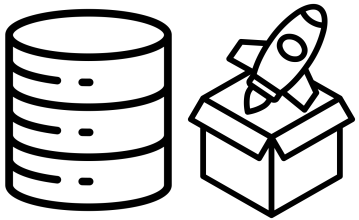
Streamlit Community Cloud



🌍 **Public deployment**

You get a public URL that anyone can visit

snowflake® Streamlit



🔒 **Private deployment**

when your app needs to stay private

What About Other Options?

👀 Other ways to deploy Streamlit apps

- Replit 💻
- Vercel ⚙️
- Netlify Drop 🌐

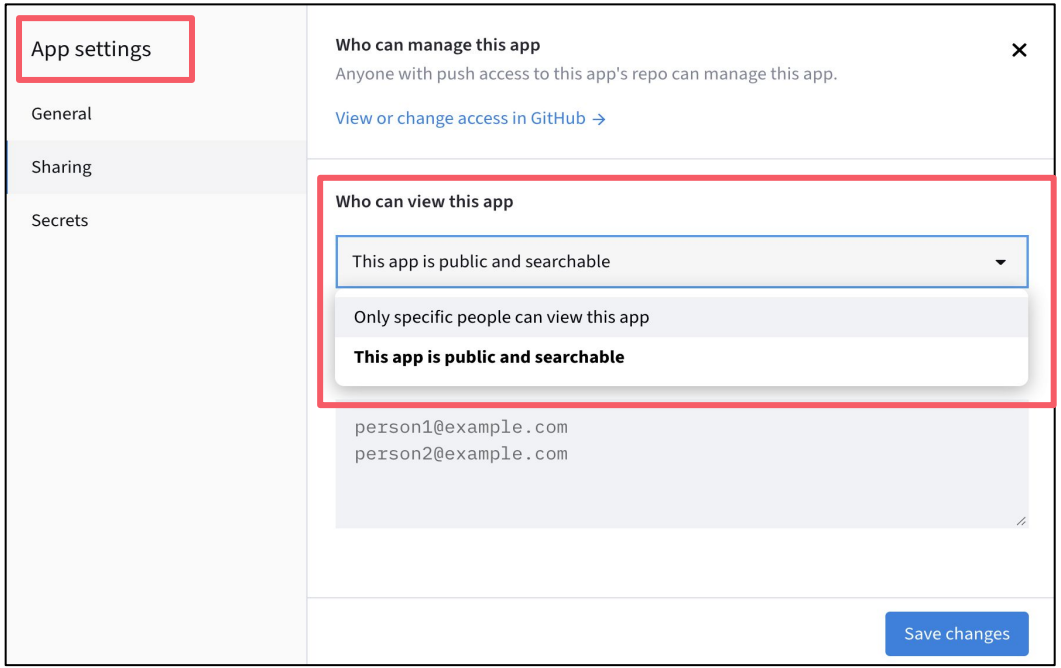
👉 Recommendation

✅ Streamlit in Snowflake

✅ Streamlit Community Cloud

- Native Python support 🐍
- Built-in security 🔐

Deploying to Streamlit Community Cloud



👤 One private app per account

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Deploying your Prototype Internally on Snowflake

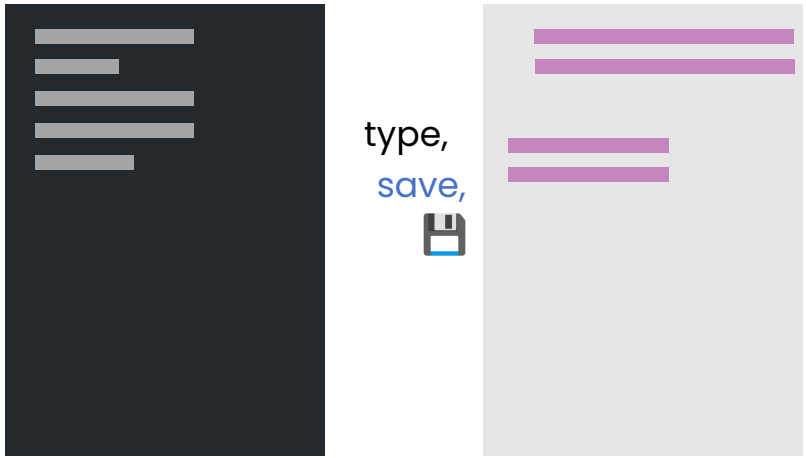
Deploying your Prototype Internally on Snowflake

Download the sentiment analysis dashboard code from the GitHub repo:

M3
└ Lesson_01
  └ deploy
    └ `streamlit_app.py`

Updating Your App

Your app uses Snowflake's built-in connection methods

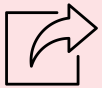


✅ Authentication

✅ Role-based access control (RBAC)

✅ Automatic session management and connection pooling for SQL queries

Role Based Access Control (RBAC)



Share your app with other users in your Snowflake account

- Access is controlled by existing Snowflake permissions and roles
- Users will need access to the underlying database and schema to view data
- The app remains secure within your Snowflake environment

Role Based Access Control (RBAC)

Share your app with other users in your Snowflake account

users with access

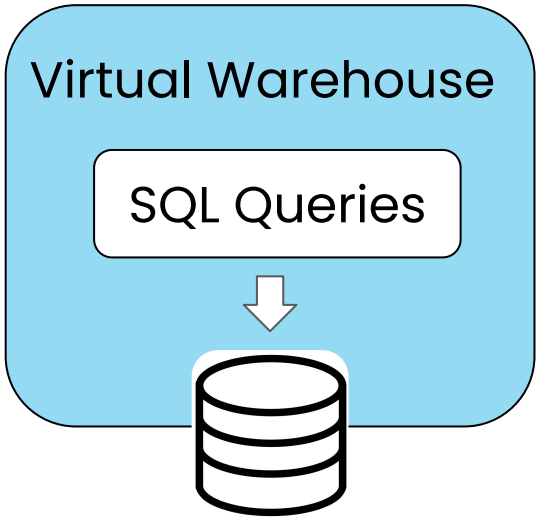
Database
└ Schema

AVALANCHE_DB
AVALANCHE_SCHEMA

✅ can view the app

- Admin Role → modify app settings and permissions.
- Data Access → follows existing Snowflake security policies.

Monitoring Your App



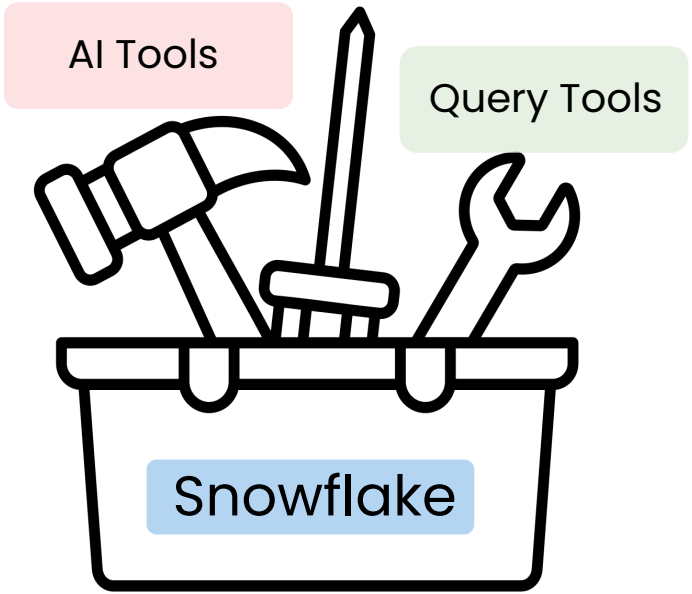
Warehouse's size impacts:

- Performance 📈
- Cost 💰

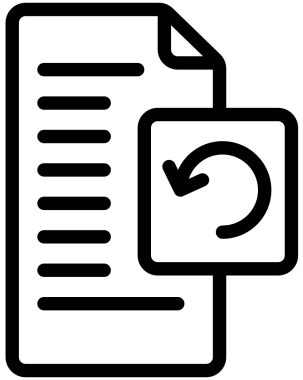
Key Metrics to Watch

- How much of your **warehouse capacity** is being used?
- How long do individual **queries** take to complete?
- The **cost** of running your warehouse over time
- How many people are **using** your app simultaneously?

Checking Query History



Activity → Query History



Checking Warehouse Usage

> Admin > Warehouses

The Warehouses Tool shows both real-time and historical warehouse performance.

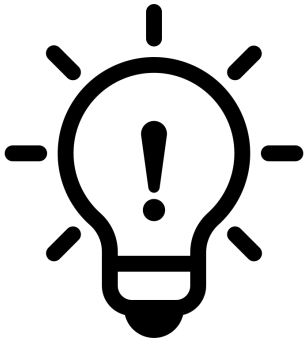
Resource Monitors → shows “cost control and usage alerts”

Practical Optimization Workflow

Weekly review process to monitor your app usage

- Check Query History
- Analyze Patterns
- Review Warehouse Usage
- Test Optimizations
- Monitor Results

Best Practices for Internal Deployment



✅ Provide clear navigation and instructions

✅ Include helpful tooltips and descriptions

✅ Ensure responsive design for different screen sizes

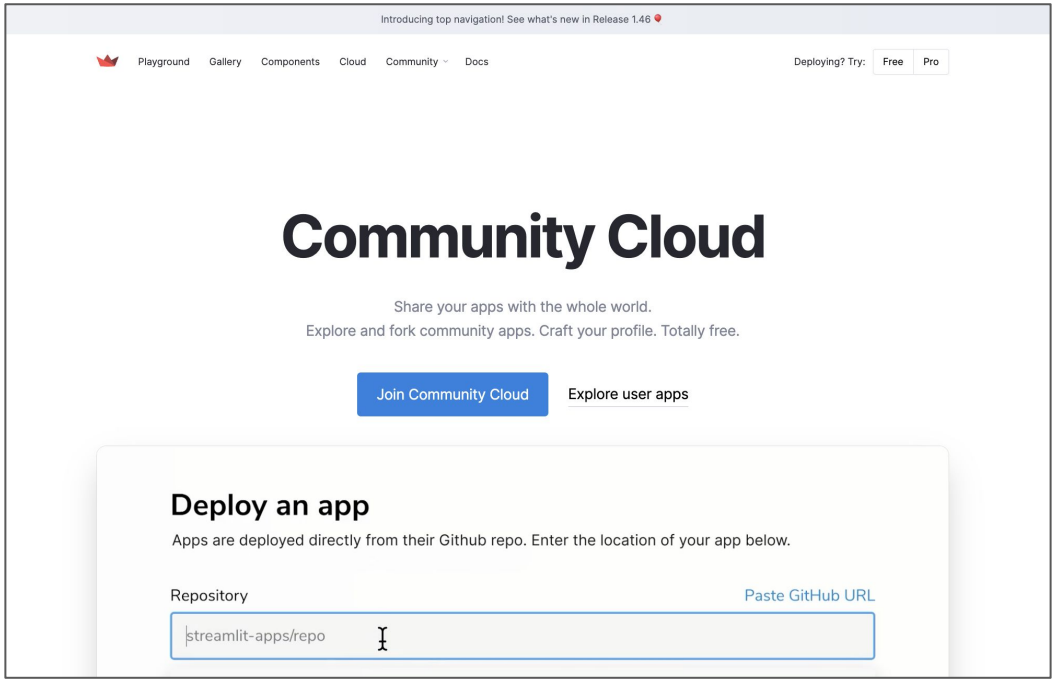
✅ Test with different user roles and permissions before rolling out

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Deploying to Streamlit Community Cloud

Join Streamlit Community Cloud



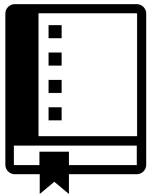
https://streamlit.io/cloud

DeepLearning.AI

snowflake®

Create Your Github Repo

1. **Make a new Github Repo**



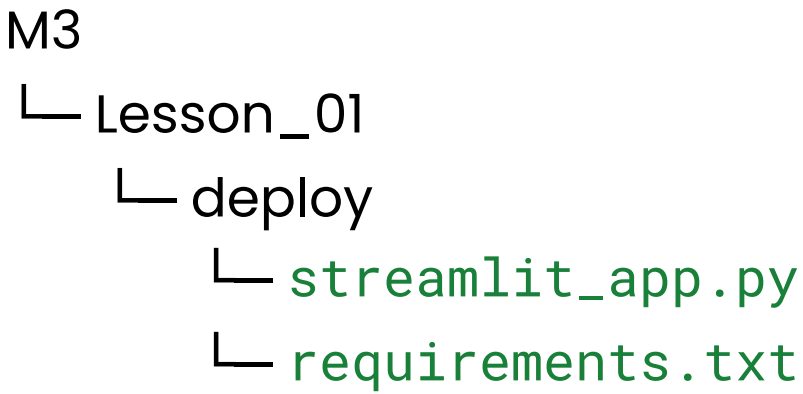
📌 **Reminder:** You need to choose a new distinct name for your repo

Upload Your Files

② **Uploading files in your new repo**

✅ In your new Github Repo, click on **"Upload an existing file"**

✅ When the upload is done, click on **"Commit changes"**



Deploy to Streamlit Community Cloud

③ **Deploy**

Log in to streamlit.io/cloud

Click “Create app”

Deploy a Public App from Github

✅ Repository, Branch: main
✅ Pointing out to the streamlit_app.py file

Choose a unique custom name

Click on “Deploy”!

Configure Snowflake Secrets

📌 Configure your credentials to access the Avalanche Snowflake database

✅ Go to your deployed app

✅ Click **"Manage App"** in the bottom-right corner

✅ Click on the three dots → and then **"Settings"**

✅ In the sidebar, click on **"Secrets"**

Configure Snowflake Secrets

```
[connections.snowflake]
account = "xxxxxxx-xxxxxxxx"
user = "your_username"
password = "xxxxxxxxxx"
role = "ACCOUNTADMIN"
warehouse = "COMPUTE_WH"
database = "AVALANCHE_DB"
schema = "AVALANCHE_SCHEMA"
```

Configure Snowflake Secrets

`https://<account_identifier>.snowflakecomputing.com`

❗ Click on Save Changes!

If Things Go Wrong

- **App won't start?**
  - Check all uploaded files into GitHub
  - Make sure your `requirements.txt` file doesn't have any typos
- **Can't connect to Snowflake?**
  - Double-check
    - account identifier
    - Username and password

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Lab 1- Deploying Your Prototype

Part 1: Internal Deployment in Snowflake

① Log into Snowsight

Snowflake > Projects > Streamlit

Click the blue "**+ Streamlit App**" button

Name your app → "Avalanche Prototype"

Select: AVALANCHE_DB and AVALANCHE_SCHEMA

Choose your compute warehouse

Click **Create**

Part 1: Internal Deployment in Snowflake

② Paste in Your Code

Editor: delete the default “Hello World” code

Copy from **streamlit_app.py** file and paste

Make sure your code connects using:

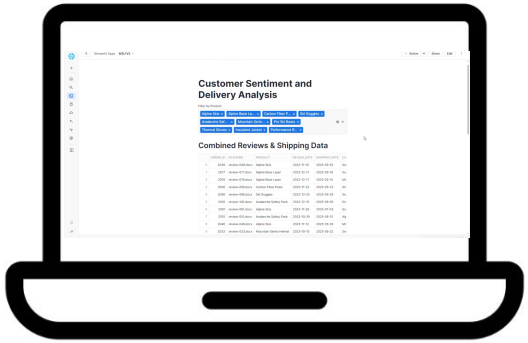
```
st.connection('snowflake')

get_active_session()
```

Part 1: Internal Deployment in Snowflake

② Run and Share Your App

Click **Run** to launch your prototype



Click **Share** to get a link you can send to teammates with access

Monitor and Optimize

✅ Check the Query History

✅ Monitor Warehouse Usage to track cost, load, and idle time

✅ Set Resource Monitors to get alerts and manage spending

💡 Pro tip:

Do a **weekly review**:

- Slow queries
- Redundant operations you can cache
- High-cost compute spikes

🧠 Bonus: Use Copilot

- Draft documentation
- Suggest code improvements
- Generate onboarding guides
- Fix performance issues

Part 2: Deploy to Streamlit Community Cloud

① **Push to GitHub**

Make your app available to the world! 🌎

Part 2: Deploy to Streamlit Community Cloud

① Push to GitHub

Create a **new public** GitHub repo

Upload the following files:

- 📄 streamlit_app.py
- 📄 requirements.txt
- 📄 streamlit/config.toml
- 📂 any relevant data folders

Part 2: Deploy to Streamlit Community Cloud

① **Push to GitHub**

```
git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin
https://github.com/your-username/avalanche.git

git push -u origin main
```

Part 2: Deploy to Streamlit Community Cloud

② Log into Streamlit Community Cloud

- Login to streamlit.io/cloud and login with GitHub
- Click **Create app** and choose the repo to deploy
- Point it to streamlit_app.py
- Click on “Deploy”!

Part 2: Deploy to Streamlit Community Cloud

App Management Menu > “Manage App”

Click on Settings > Click on Secrets

② **Log into Streamlit Community Cloud**

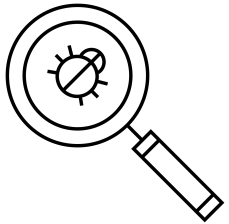
```
[connections.snowflake]
account = "your-account-id"
user = "your_username"
password = "your_password"
role = "ACCOUNTADMIN"
warehouse = "COMPUTE_WH"
database = "AVALANCHE_DB"
schema = "PUBLIC"
```

Click on Save

Test and Monitor

Visit the website address for your app:
https://your-app-name.streamlit.app

Settings menu >
click on **Manage App** >
Logs to check for any issues



Lesson 2

Getting Feedback

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Iterate Quickly

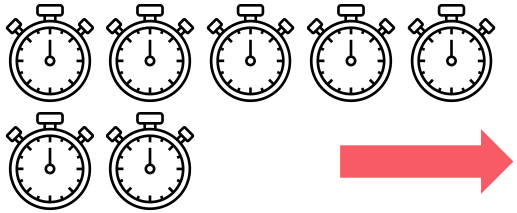
Narrow Your Scope

With AI, you can build things faster than ever before!

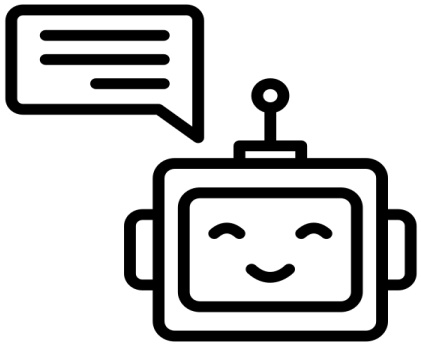
Build It Quick and Dirty, Then Fix It

Traditional programming

GenAI programming



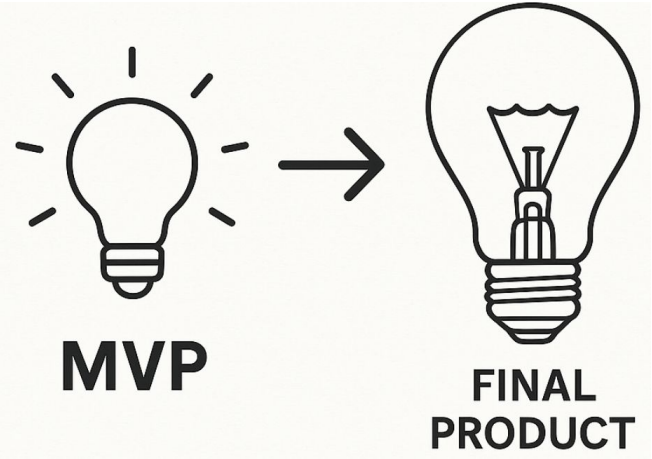
uncover problems by programming



- Build a basic version of the chatbot
  - Answer the five common questions
- See where users get confused or frustrated
- Improve and repeat

Remember, It's Just a Prototype

You're building something simple to test your ideas

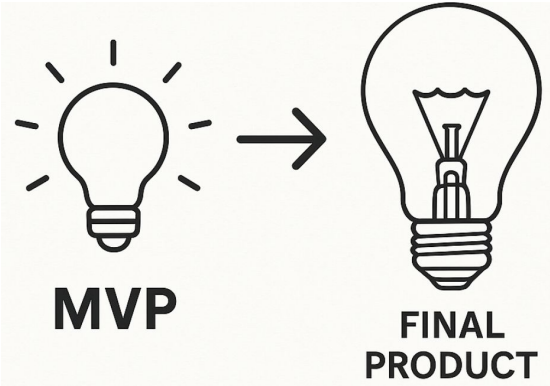


✅ The simplest version

Know When to Stop

When is your prototype "**good enough**"?

Done is better than perfect!



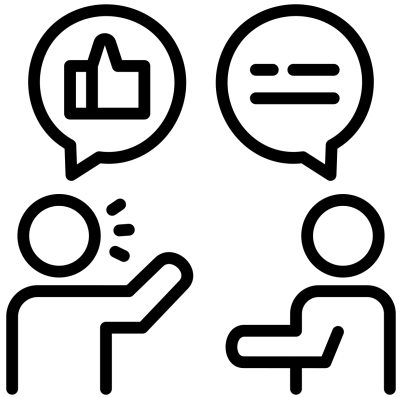
Your prototype just needs to:

✅ work well enough to test your main ideas

✅ get feedback

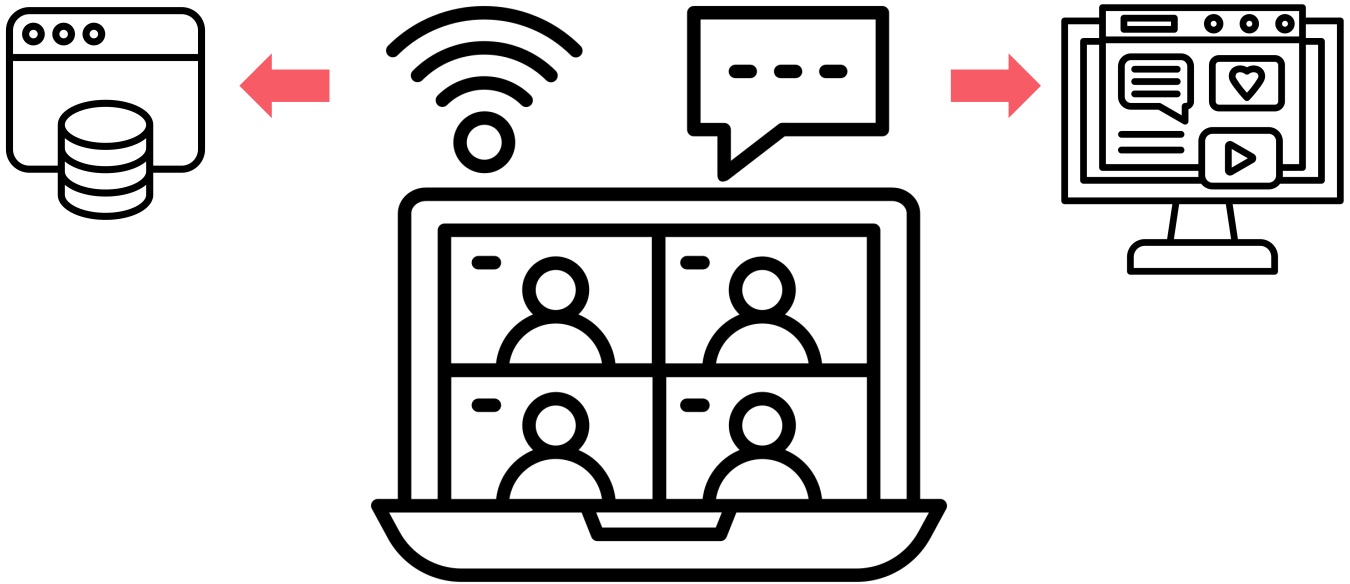
Deliver 80% of the core functionality, **efficiently**

Close The Loop With Users

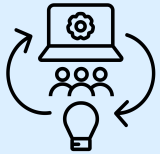


- ✅ Tell people what you changed based on their feedback
- ✅ Send them updates
- ✅ Write release notes
- ✅ Show them in the app

Share What You Learned



Keep Improving



Development isn't a one-time thing

The people who succeed are the ones that keep listening and keep improving

Your competitors are getting better too, so you need to stay ahead

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

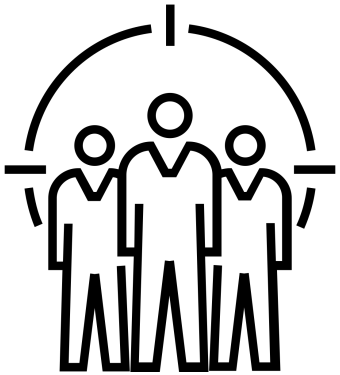
Fast Feedback

Not All Feedback Is Created Equal

Early-stage AI development → qualitative feedback

Collect feedback efficiently and strategically

Start Small and Start Real

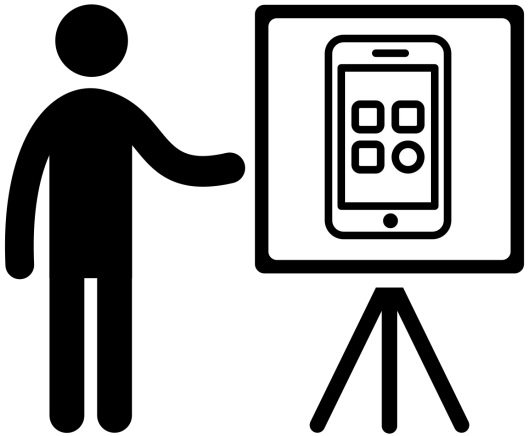


target user group

🔎 Find tester users:

- Social media
- Professional networks
- Existing customer base

The 5 Second Rule



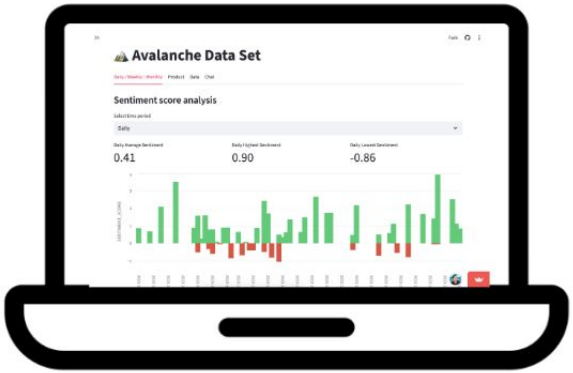
Ask them:

- What do they remember?
- What do they think the app is supposed to do?

Ask at the Right Moment

Before Use	During Use	🎯 **After Use**
Low feedback quality	Distracted, partial feedback	Best time — fresh, clear experience

The One True Path



⚠️ **Important →**
Perform a sentiment analysis on an input dataset

Ask your testers to:

- Select a date range and set of products, then review the results

Ask Better Questions

Broad & vague

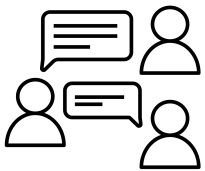
🚫 "What do you think?"

🚫 “Do you like it?”

Specific & actionable

✅ "Was this button easy to find?"

✅ "How did this feature help you solve your problem?"

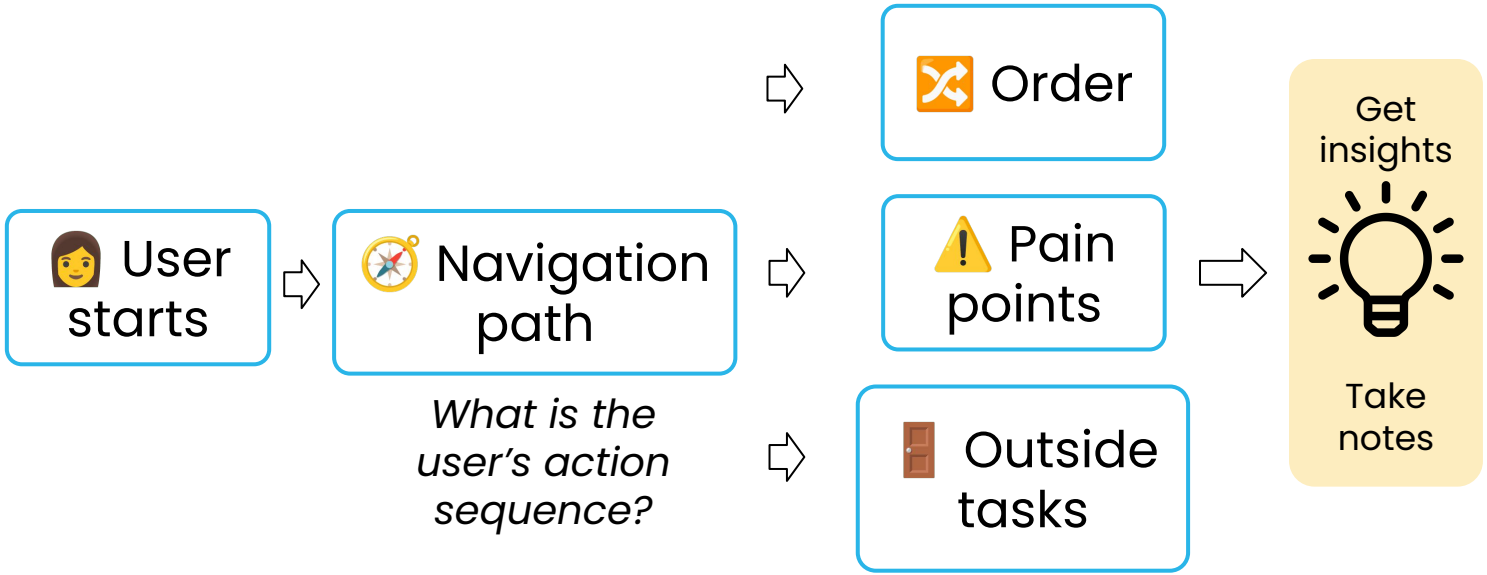


testers

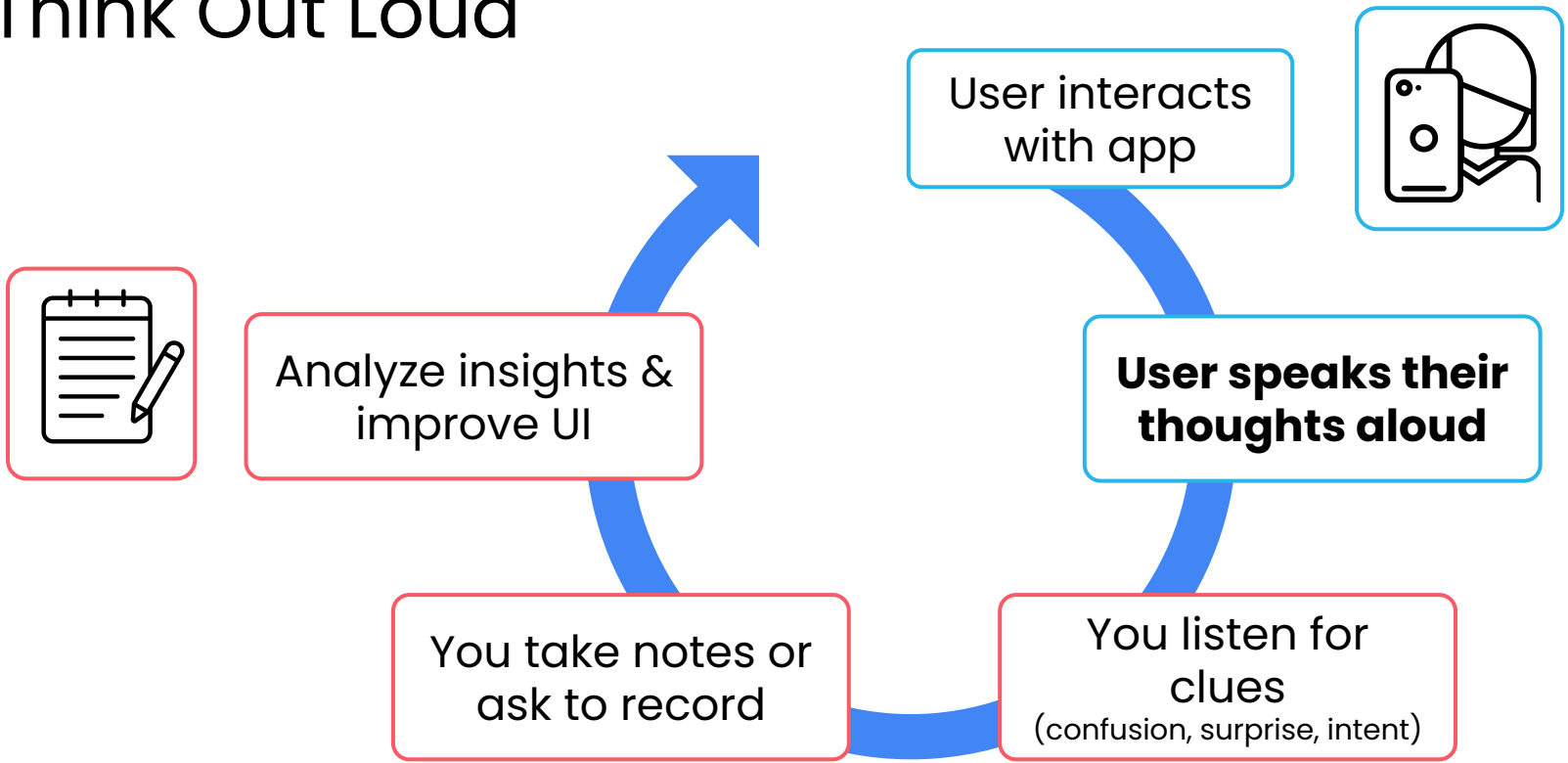
Ask open-ended questions → 🧠 Insights

Ask open-ended questions → 📊 Trends

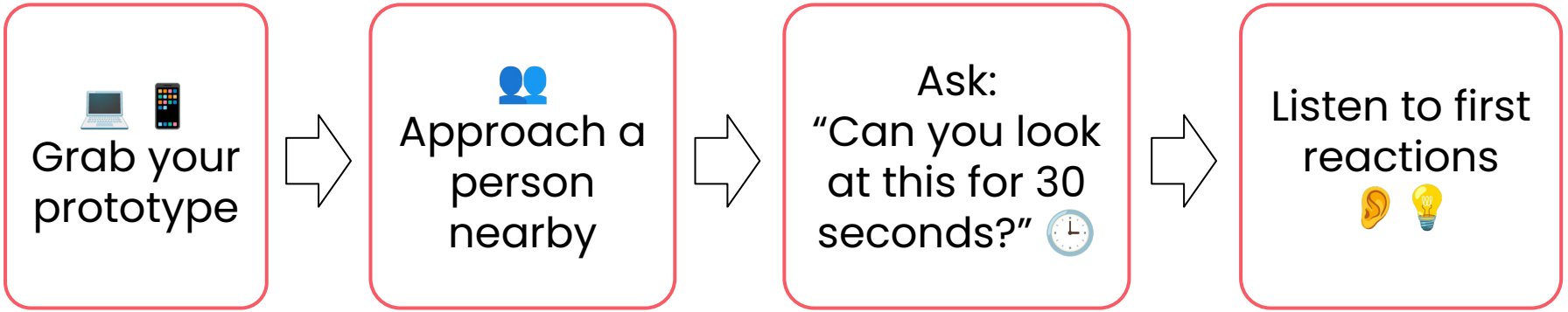
Channel Your Inner Scientist



Think Out Loud



Hallway Feedback



Wrap Up

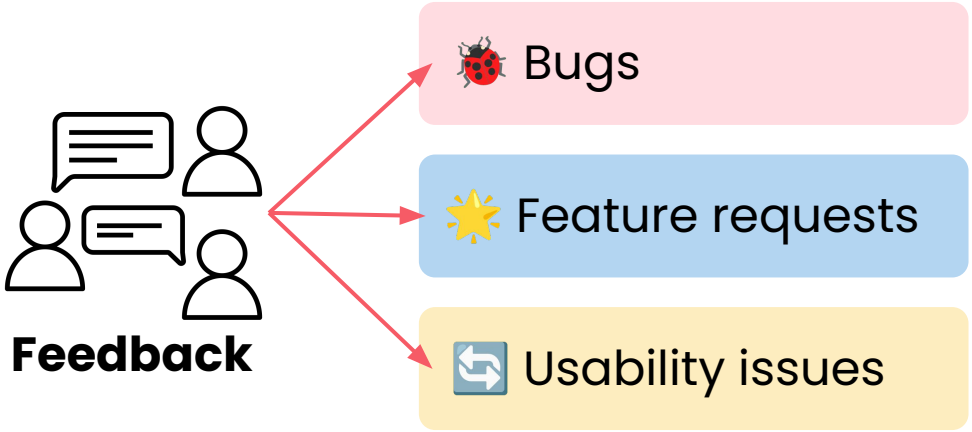
- Gather feedback early and often 📥
- Prioritize feedback using the impact vs. effort matrix ✍️
- Focus on high-impact, low-effort changes first 🔎

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Acting on Feedback

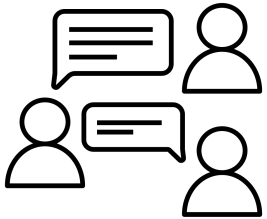
Organize Feedback Into Buckets



✅ WHAT people are asking for

✅ WHY they are asking for it

Think Like a Detective



Feedback

"your app is slow"

"this feature doesn't work”

🔎 Think like a detective

Break the problem into slices

- Is it slow for new users or returning users?
- Is it affecting mobile users more than desktop users?

Measure Your Changes

Your measurement approach should be just as fast as your building

🎯 Quick validation methods

- 👀 User observation
- 📬 Micro-survey
- 📈 Basic analytics check
- 🆚 A/B test (simple)

Use AI To Speed Everything Up

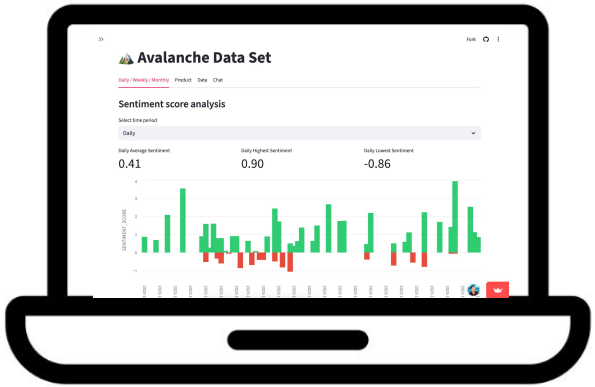
Let AI help you code

Get good at prompting

Try agentic workflows

Track feedback systematically

Acting on Feedback



Some comments:

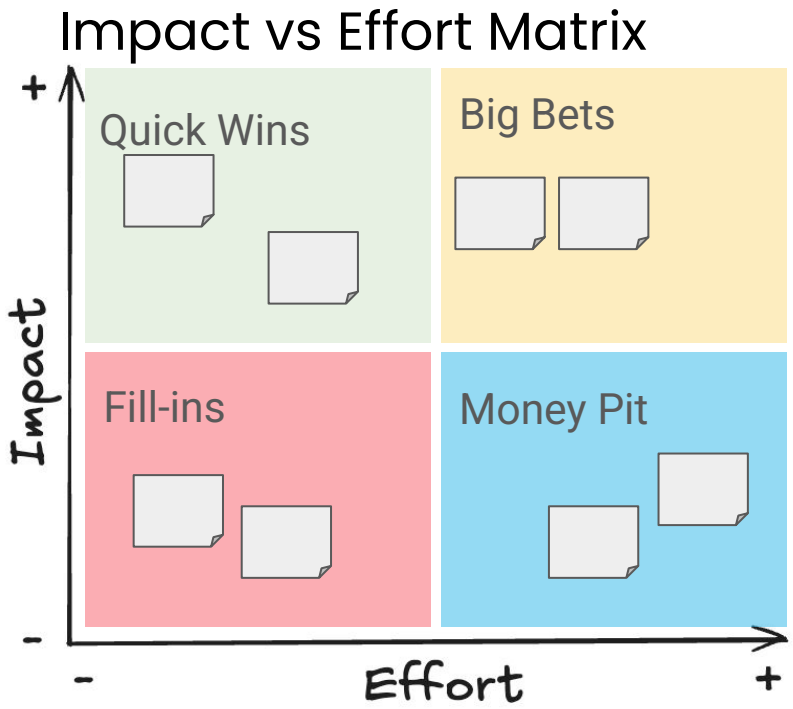
I asked it questions about the dataset, and it gave me incorrect answers

It didn't really seem to know much about our company's research methodology.

It would be great if we could choose other datasets to analyze.

Feedback Driven Prototyping

💡 Remember the
"Must Have"
"Nice to Have"
decision framework
→ MVP Building Plan

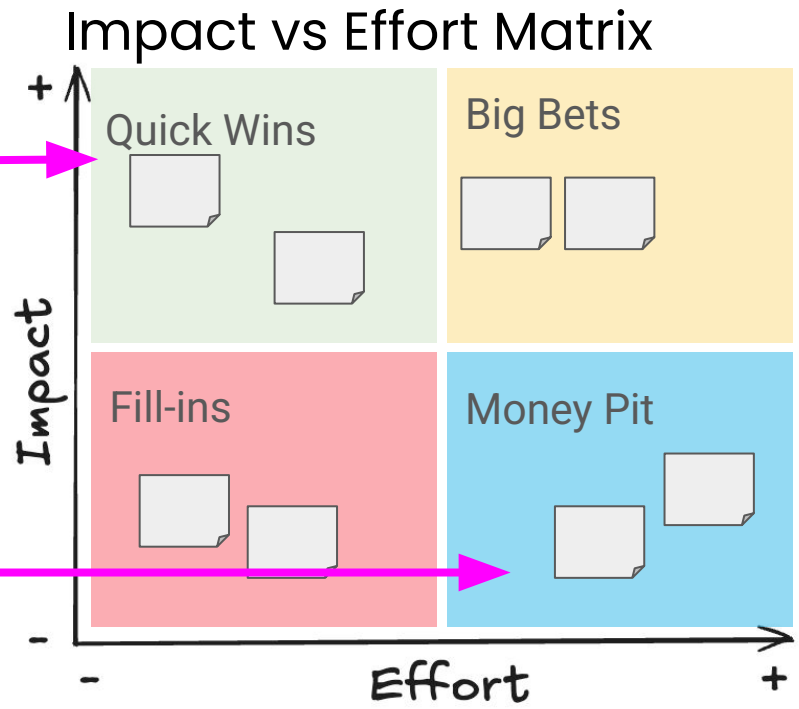


Feedback Driven Prototyping

I asked it questions about the dataset, and it gave me incorrect answers

It didn't really seem to know much about our company's research methodology.

It would be great if we could choose other datasets to analyze.



DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Iterate, Improve, Repeat – Fast Feedback for Your Avalanche App

By The End of This Lesson

1. Test your app from a user's perspective
2. Collect actionable feedback quickly
3. Prioritize improvements based on impact and effort
4. Use AI tools to accelerate your iteration process

The Scenario

Open your deployed Avalanche app:

- Snowflake's Streamlit interface
- Streamlit Cloud (if you deployed there)

What do customers think about our winter product line?

Step 1: Critical User Journey Test

✅ Evaluate:

- Can you quickly locate the sentiment analysis feature?
- Are the results presented in an easy-to-understand format?
- Would someone unfamiliar with your dataset know what to click next?

Step 2: Run a 5-Second Test

❓ Ask:

- What do you think this app does?
- What stood out to you most?
- Could you figure out what to do next?

Step 3: Test the “One True Path”

User’s journey:

- Load the dashboard
- Run sentiment analysis
- Interpret the results

❓ Ask yourself:

- Is anything confusing or awkward in this flow?
- Are instructions clear and actionable?
- Does the layout support this main task effectively?

Step 4: Organize Your Feedback

🐞 Bugs

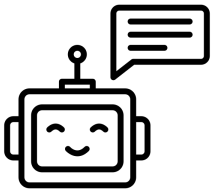
🔄 Usability Issues

🌟 Feature Requests

❓ Ask:

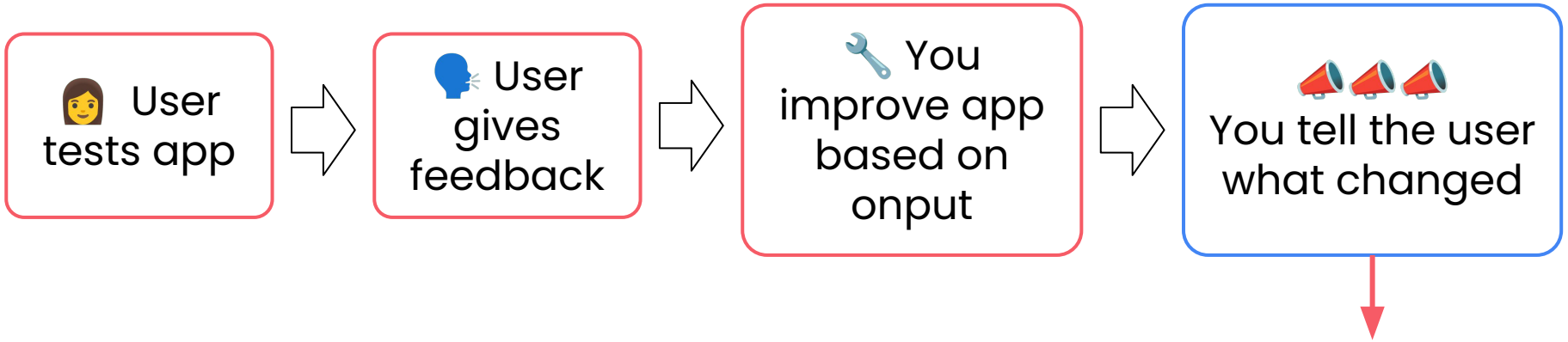
- What's high-impact but low-effort to fix?
- What can I improve right now in under 30 minutes?

Step 5: Use AI to Accelerate Improvements



Tools	Content Issues	UX Problems
GitHub Copilot, ChatGPT, Claude Ask AI?	rephrase confusing labels or tooltips	**GenAI Assistant tab in Snowflake** → test new ways of answering user questions
	rewrite prompt templates for better clarity	suggest alternative chart types or data visualizations
	cleaning up code blocks	

Step 6: Close the Loop



Lesson 3

Improving Your Results

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Improving Prompts

Why Prompt Engineering Matters for Fast Prototyping

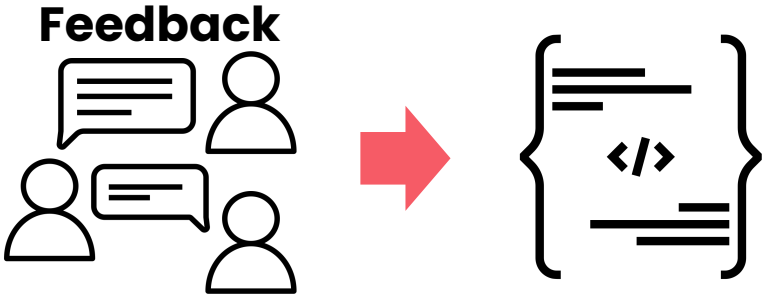
In fast prototyping, time is everything

- ✅ Build a working prototype quickly
- ✅ Test it with real users
- ✅ Gather feedback
- ✅ Implement changes fast
- ✅ Repeat the cycle

❌ Poor prompts slow you down

✅ A clear, specific prompt gets you 95%+ of the way

The Feedback Integration Problem



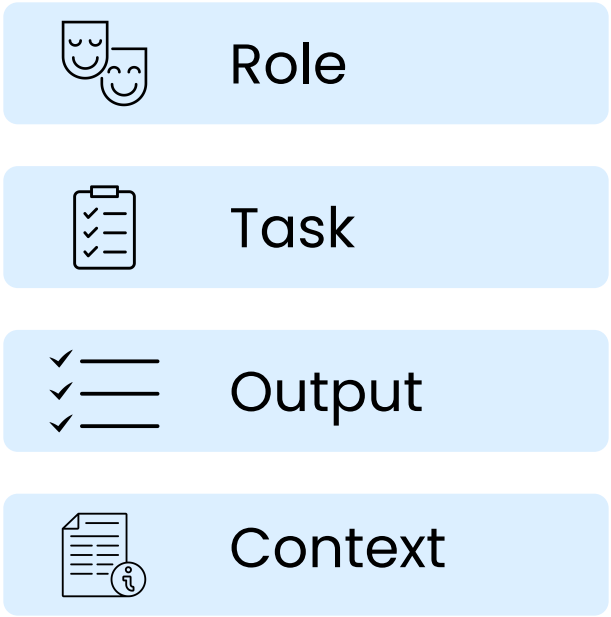
- "the chart is confusing"
- "I can't find the export button"

Try prompting your AI like this:

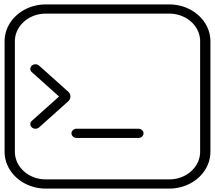
You are a UX-focused Python developer. The user said "the chart is confusing."

Rewrite this chart code to include:
- Clear axis labels
- A legend that explains what each line represents
- Hover tooltips showing exact values
- A title that explains what the chart shows

The Four Building Blocks of Effective Prompts

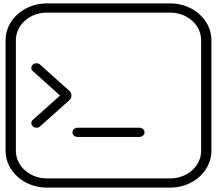


Instructions: Tell the AI Exactly What to Do



Make an app

⏳ hours modifying generic code



"Build a Streamlit app that analyzes PDF research papers and answers questions about them"

Context: Provide Essential Background

🚫 Without context: AI might make wrong assumptions

✅ Add context for speed & precision

Role setting → "You are a Python developer experienced in Streamlit apps"

Constraints and requirements → Specify...
- Python versions
- deployment platforms
- technical limitations

Input Data: Supply the Raw Materials

🚫 Vague input → Generic output

✅ Real Input → Real solutions

Examples of concrete input:

- Specific code snippets to modify or extend
- Example data formats (e.g., CSV, JSON)
- Sample files or datasets

Output Indicators: Specify the Response Format

❗ Problem: Unclear output = Wasted time

✅ Solution: Be explicit about the format you want

💬 **Say things like:**

- `"Return only working Python code"`
- `"Format the output as JSON"`
- `"Give me a single Python file"`

🔄 **When iterating fast:**

- `"Return only the modified function"`
- `"Show just the CSS changes"`

Prompt Engineering Strategies for Fast Prototyping

1 Start simple, then iterate

- Get something working quickly
- Improve it based on user feedback
- Don't try to build everything at once
- Create a basic version first, then refine it step by step

Benefits:

✅ Better control

✅ Reliable improvements

✅ Fast feedback cycles

Prompt Engineering Strategies for Fast Prototyping

② Break complex tasks into smaller prompts

- Basic app structure
- Add a specific feature
- Handle errors
- Optimize performance

Example:
🗣️ User: "App is slow & charts are confusing"

✅ Prompt 1:
"Optimize this data loading code for better performance"

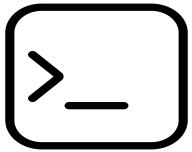
✅ Prompt 2:
"Improve these chart visualizations for better clarity"

Use Guardrails for Consistency

Guardrails keep your code consistent and maintainable

Guardrail	Why It Helps
Tell the AI what to avoid	Prevents unwanted complexity or breaking changes
Ask for reasoning	Understand why changes are made — stay in control
Use consistent terminology	Keeps codebase coherent across iterations
Maintain your coding standards	Code stays clean and readable, even under pressure

A Real World Example

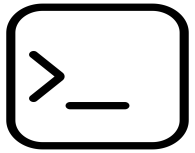


You are a Python developer experienced in Streamlit apps.

Build a PDF research paper analyzer with these features:
- Upload PDF functionality
- Extract and display title and abstract
- Summarize main contributions (under 300 words)
- Interactive Q&A using GPT-4

Assume PyMuPDF and Streamlit are installed.
Return only complete working code in a single Python file.

The Role Setting

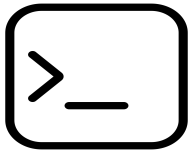


You are a Python developer experienced in Streamlit apps.

Build a PDF research paper analyzer with these features:
- Upload PDF functionality
- Extract and display title and abstract
- Summarize main contributions (under 300 words)
- Interactive Q&A using GPT-4

Assume PyMuPDF and Streamlit are installed.
Return only complete working code in a single Python file.

Clear Task + Features

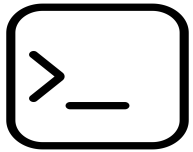


You are a Python developer experienced in Streamlit apps.

Build a PDF research paper analyzer with these features:
- Upload PDF functionality
- Extract and display title and abstract
- Summarize main contributions (under 300 words)
- Interactive Q&A using GPT-4

Assume PyMuPDF and Streamlit are installed.
Return only complete working code in a single Python file.

The Technical Assumptions

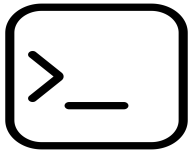


You are a Python developer experienced in Streamlit apps.

Build a PDF research paper analyzer with these features:
- Upload PDF functionality
- Extract and display title and abstract
- Summarize main contributions (under 300 words)
- Interactive Q&A using GPT-4

Assume PyMuPDF and Streamlit are installed.
Return only complete working code in a single Python file.

The Output Specification

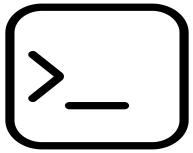


You are a Python developer experienced in Streamlit apps.

Build a PDF research paper analyzer with these features:
- Upload PDF functionality
- Extract and display title and abstract
- Summarize main contributions (under 300 words)
- Interactive Q&A using GPT-4

Assume PyMuPDF and Streamlit are installed.
Return only complete working code in a single Python file.

Refining Apps with GenAI Feedback



You are a UX-focused Python developer. Based on user feedback that the dashboard is "too cluttered" and they "can't find the export button," modify this Streamlit code to:

- Move the export button to a prominent location in the sidebar
- Reduce visual clutter by using expandable sections for secondary features
- Add clear section headers to organize the content
- Ensure the most important metrics are visible at the top

Return only the modified code sections.

Review and Enhancement Process

Systematic approach to keep improving your prototype:

- Rapid Code Review 🔍
- Performance Optimization ⚡
- Feature Expansion 🧩
- UX Enhancements 🎨

Prompt → Prototype → Feedback → Improvement → Repeat

DeepLearning.AI

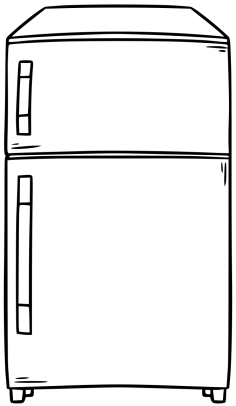
Fast Prototyping of GenAI Apps with Streamlit

Upgrading Your Prototype with Data Augmentation

What is Prompt Augmentation?

Add specific information about your dataset right into the prompt

Give your prototype → a cheat sheet



"Guess what's in my fridge."

"Here's what's in the fridge — now tell me what I can cook"

Gen AI Chatbot and Avalanche

🗣️: “What products are in the Avalanche dataset?”

🤖: “I don’t have access to specific product information.”

*Currently, the chatbot knows nothing about the Avalanche dataset ☹️

💡 Solution

Feed the Data into the Prompt

Getting Set Up

Starting Point:

➡️ **Located in the GitHub repo:** M3/Lesson_03/

DeepLearning.AI

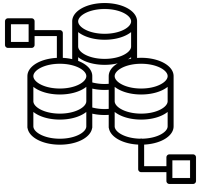
Fast Prototyping of GenAI Apps with Streamlit

Using RAG to Improve Model Performance

What is RAG?

RAG: Retrieval-Augmented Generation

User query



Retrieval system

- Search docs or DB
- Find top relevant info

Generation

- Input info + query
- LLM writes a response

Final answer

Why Does This Matter?

🚫 **LLMs don't know everything**

- Don't know your latest product data
- Can't access internal documents or company policies
- Aren't reliable in niche domains (e.g., medicine, law)

✅ **RAG = smarter responses without retraining**

- Pulls info at runtime
- Ideal for accuracy, transparency, and trust
- Great for apps in:

  🏥 medical research, ⚖️ contract law, 📊 enterprise data

When Use RAG Instead of Just Prompting?

⚠️ Challenge	💡 RAG Solution
Your knowledge base is too large to fit in a prompt	→ Retrieve only the most relevant info
Your content changes frequently	→ Pulls real-time data at the moment
You want source attribution	→ Keeps links or citations
Working in specialized field	→ Boosts accuracy
You want personalized responses	→ Fetches user-specific data on the fly
You're optimizing token usage	→ Smaller prompts = lower cost

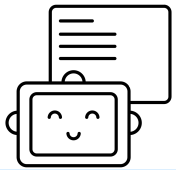
Let’s Bring it Back to the Avalanche prototype

User question

“What are customers saying about our return policy this month?”

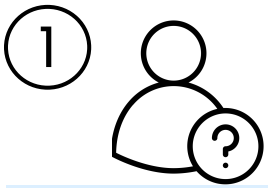
RAG process

- **Retrieval step**
  🔎 Search Avalanche reviews from July
  → Finds comments mentioning "return policy"
- **Generation step**
  🧠 LLM writes a response using real customer quotes



Answer

Working Behind The Scenes



User question

2 Break data into chunks

3 Retrieve relevant chunks

4 Feed chunks + question to LLM

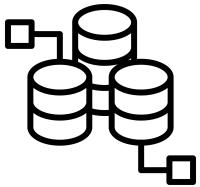
5 Generate grounded answer

Snowflake Cortex

https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-search/cortex-search-overview

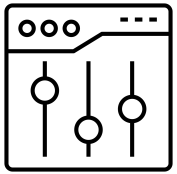
What About Fine-tuning? How is RAG Different?

RAG



New context at query time

Fine-Tuning



Behavior into the model itself

- Need a specific writing style or tone
- Doing the same task over and over
- Need consistent logic or reasoning
- Don’t want to manage external documents or storage

Most Production Systems Use a Hybrid Approach

Prompt engineering	⇨	guide the model step-by-step
RAG	⇨	inject relevant, factual context
Fine-tuning	⇨	lock in style, behavior, or reasoning for repetitive tasks

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Setting up a RAG Pipeline Using Cortex Search

Setting Up a RAG Pipeline Using Cortex Search

- Cortex Search is Snowflake's managed search service for AI applications
  - 🔍 Automatic indexing & embeddings
  - 🚀 Simple APIs for RAG applications
  - ⚙️ No infrastructure management required

Setting Up a RAG Pipeline Using Cortex Search

Follow along by:

- Opening a new Snowflake notebook
- Or finding the code in the course GitHub repo:
  - M3/Lesson_03/Lab2

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

Lab 2- Integrating RAG into Your Chatbot

You Did It!

✅ From basic Streamlit to advanced RAG integration.

✅ You can quickly prototype any data app idea, add GenAI capabilities, and turn concepts into working solutions.

DeepLearning.AI

Fast Prototyping of GenAI Apps with Streamlit

What Comes Next?

What You've Learned

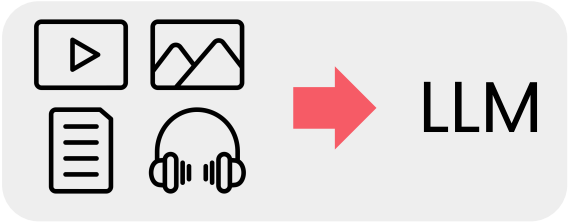
- From ingestion → to analysis → to UX
- Using real-world data
- Streamlit to build polished front ends, fast
- Snowflake for powerful data handling
- Cortex to supercharge it all with GenAI

idea

Working prototype

What Comes Next?

1. **Build a multi-modal app**



"You record a screencast of your app in use..."

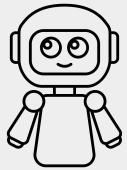
Feed it to a model...

Ask for

- UX feedback
- Documentation
- Next-step features — all automatically

What Comes Next?

② **Prototype with AI agents**



agentic AI

Create custom agents that behave like junior developers

Give:

- persona
- some goals
- dev style ...

MVP-first approach

What Comes Next?

③ Turn your prototype into a product

For work,
a side project,
or just for fun

idea → working GenAI app

prototype → product

- Add authentication and permissions
- Integrate with external APIs
- Turn insights into features
- Deploy and monetize

The Mindset That Matters

- AI will keep evolving 🔥
- The tools will change 🔧
- The interfaces will change 💻
- The models will change 📊

🤩

Mindset!

Ideate. Build. Evaluate. Iterate. Improve. Repeat.

Concluding Remarks

Got the skills to build smarter apps — faster:

- Keep prototyping
- Keep experimenting
- Keep letting GenAI amplify what you bring to the table

Congratulations!