DeltaForceOS

CODEX ROUTE / INTERMEDIATE MODULE 01

Evals for Agent Builders: Codex Route

BUILD OUTCOME

Create an eval harness so prompt changes and model swaps do not silently break production.

BEST FOR

Use this PDF when you want exact prompts and clicks for this route instead of general theory.

MODEL TIER

Light for exact checks; mid for rubric grading; heavy only for disputed examples.

FIRST INSTRUCTION

```
Inspect this repo for the Evals for Agent Builders build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

Before using the agent, make the workspace boring and clear.

01 Open the project folder. If you do not have one yet, create a new folder on Desktop and name it after the lesson.

02 Install Node.js if npm is not available. Beginners can check by opening Terminal and typing npm -v.

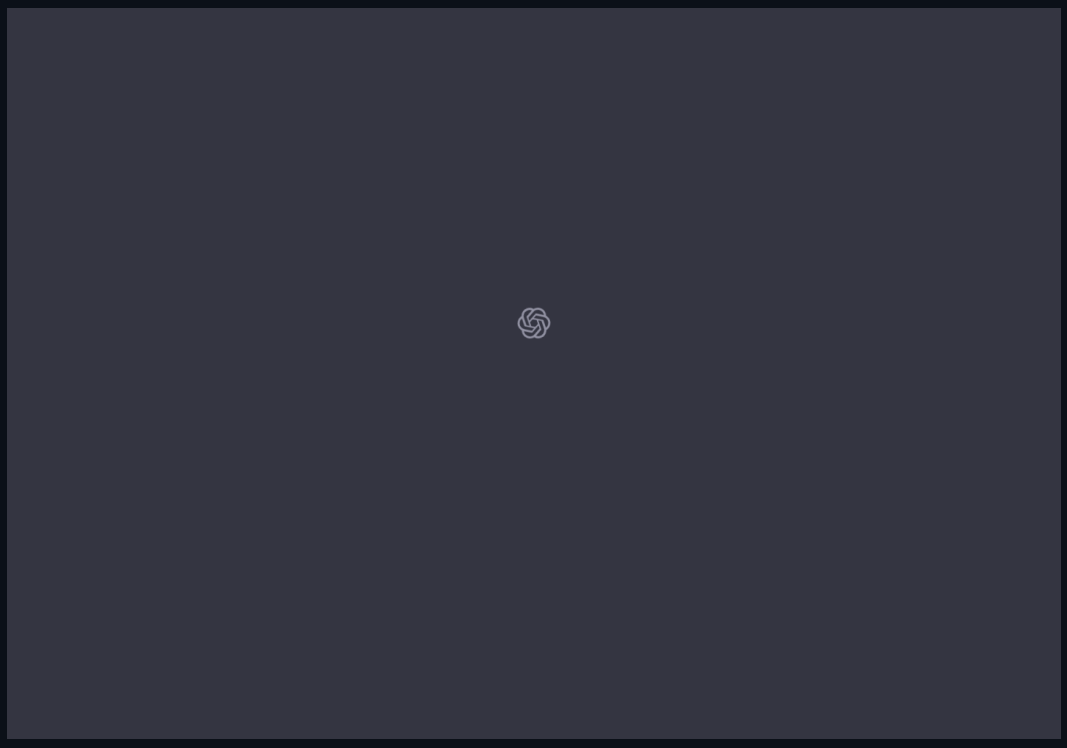
03 Create .env.local in the project root and add only the keys this lesson needs.

04 Confirm .env.local is listed in .gitignore so secrets do not go to GitHub.

05 Open the official dashboard for the first two tools in this lesson and create one tiny test project.

06 Decide where proof will live: Supabase row, screenshot, live URL, or Skool post.

OpenAI: click path for this route.



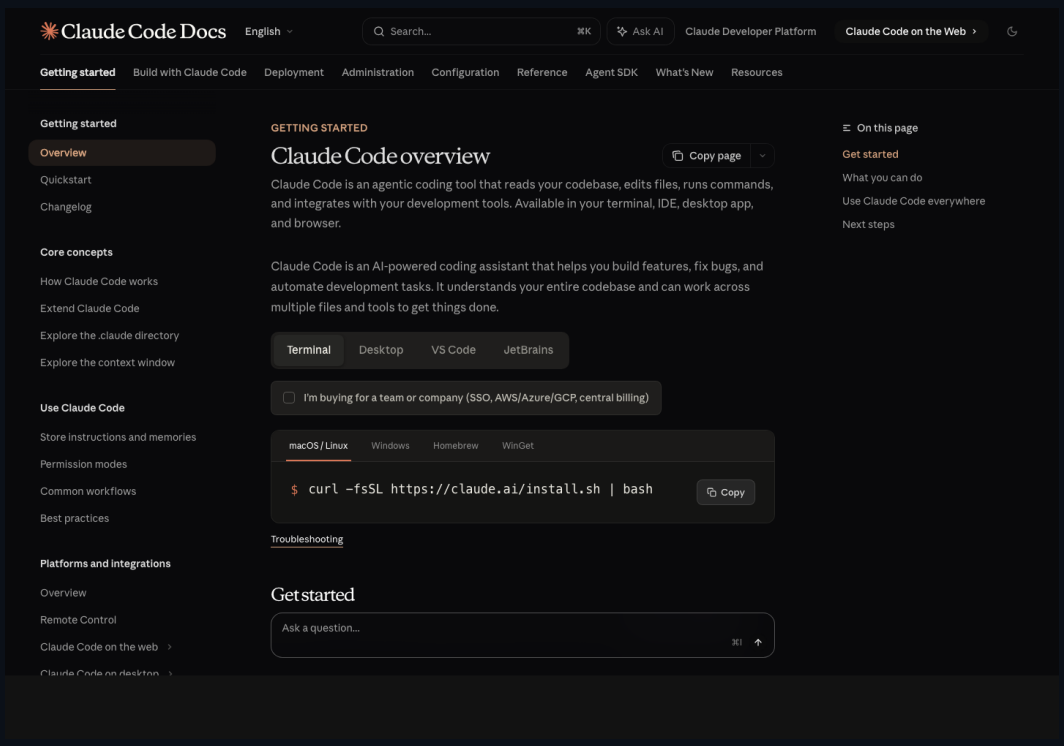
WHERE TO CLICK

Open the OpenAI platform, create an API key, add billing limits, and start with one small Responses API call.

ENV NAMES

```
OPENAI_API_KEY
```

Claude: click path for this route.



WHERE TO CLICK

Open Anthropic Console, create a key, and use Claude for planning, review, and long-context reasoning.

ENV NAMES

```
ANTHROPIC_API_KEY
```

Copy this box exactly, then let the agent ask clarifying questions only if needed.

INSPECT PROMPT

```
Inspect this repo for the Evals for Agent Builders build.
Explain the files a beginner needs: README.md, package.json, src, public, scripts, .env.local, and any
Supabase files.
Do not edit yet. Give me the smallest safe build plan.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

Copy this box exactly, then let the agent ask clarifying questions only if needed.

BUILD PROMPT

```
Implement the smallest useful version of Evals for Agent Builders.
Outcome: Create an eval harness so prompt changes and model swaps do not silently break production.
Tools allowed: OpenAI, Claude, Supabase, PostHog, GitHub, TypeScript, Python, Codex
Keep secrets server-side. Add code comments only where a beginner would get lost.
```

BEGINNER CHECK

If the agent proposes a huge rebuild, ask it to shrink the scope to the smallest artifact that proves this lesson.

DO NOT PASTE

Do not paste service-role keys, payment secrets, signing keys, or private client data into the chat.

TYPE THIS INTO FILES

```
# README.md lesson block
Lesson: Evals for Agent Builders
Outcome: Create an eval harness so prompt changes and model swaps do not silently break production.
Tools: OpenAI, Claude, Supabase, PostHog, GitHub, TypeScript, Python, Codex

# .env.local placeholders
NEXT_PUBLIC_SUPABASE_URL=your_value_here
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_value_here
OPENAI_API_KEY=server_only
ANTHROPIC_API_KEY=server_only

# Verify
npm run build
```

WHERE IT GOES

README.md gets instructions. .env.local gets real local values. Vercel gets production values. Source files get implementation.

Basic English. Click by click. Do not skip ahead.

01 Open Supabase Table editor in your project. Click + New table named eval_cases with columns: id (uuid), name (text), input (text), expected_output (jsonb), risk_level (text default 'low').

02 Click + New table again named eval_runs with columns: id (uuid), case_id (uuid foreign key to eval_cases), prompt_version (text), model (text), actual_output (jsonb), passed (bool), cost (numeric), created_at (timestamp default now()).

03 Open Supabase Table editor and the agent_runs table from earlier modules. Pick ten real outputs (mix of good and bad).

04 For each, write the expected_output JSON in plain English. Insert all ten rows into eval_cases via the table editor.

05 Mark three of them with risk_level='high' (the ones that affect users or money).

06 In your editor, create src/lib/evals.ts. Write an async function runEvals(promptVersion, model) that selects all eval_cases.

Finish the build. The last step always posts proof in Skool.

01 Inside runEvals, loop over each case. Call your model with case.input and the prompt for promptVersion. Compare the result to case.expected_output using JSON.stringify equality first.

02 Insert one row into eval_runs for each case with passed true or false, the actual_output, cost, and latency.

03 In Terminal, run: npx tsx src/lib/evals.ts (or wire it to a Next.js API route at /api/evals). Watch the console output.

04 Open Supabase and the eval_runs table. Count passed rows. Note total cost and average latency.

05 Now change one prompt in your code. Run runEvals again with the new promptVersion. Compare passed rate and cost side by side.

06 Promote the new prompt only if passed rate goes UP without cost going up too much. Save the comparison screenshot for the Intermediate 01 Skool thread.

The build is not finished until verification is boring.

01 Does the eval set include failures from real users?

02 Can you compare two model versions on the same cases?

03 Do failures block production or just create a warning?

04 Run npm run build or the focused test command.

05 Download or open the produced artifact and inspect it visually.

06 Write the failure and fix in README.md or the Skool proof post.

01 Commit safe files to GitHub after reviewing the diff.

02 Open Vercel and deploy a preview first.

03 Add missing env vars under Project Settings -> Environment Variables if the build fails.

04 Run the workflow with fake or test data in preview.

05 Promote or deploy production only after the smoke test passes.

06 Save the live URL and proof artifact in Skool.

AGENT DEPLOY PROMPT

```
Verify the preview build, list any missing env vars, and give me the exact smoke test before production.
```

If something goes wrong, ask for a small repair instead of a total rewrite.

01 Testing only easy examples.

02 Letting subjective quality become untracked opinion.

03 Optimizing cost while quietly losing accuracy.

04 Ask the agent to explain the failing layer: local app, env var, database, external API, prompt, deploy, or auth.

05 Ask for one fix, one test, and one rollback step.

06 Do not let the agent change unrelated files to hide the failure.

SKOOL PROOF

Create ten eval cases for your first agent and run a before/after prompt comparison.

WHAT TO POST

Artifact, screenshot or URL, what worked, what broke, exact prompt used, and the next target.

SOURCES

Codex: https://developers.openai.com/codex

OpenAI: https://platform.openai.com/docs/quickstart

Claude: https://docs.anthropic.com/en/docs/overview

Supabase: https://supabase.com/docs/guides/getting-started

PostHog: https://posthog.com/docs

GitHub: https://docs.github.com/en/get-started