Setup Codex with GPT-5

Requirements

As everything in the Group, CLI connectivty is dependant on having the right proxy and SSL settings.
Sounds easy, it isn't... To solve proxy issues, you should run Alpaca:
🦙Setting up Alpaca on MacOS or 🦙Setting up Alpaca on Windows
🤯CLI - Why you should not set CNTLM nor Prisma
To solve SSL issues, you should build a PEM cert store and point your CLI to it using various environment variables...
🔐Group Internal Root CAs Certificate Store

Or if you're `on MacOS,` you can head there and let my script do it for you: Proxy and SSL Certificate issues

Request `AR - GenAI Studio - Prod - User` access in identity.cba to access it at: https://studio.genai.cba and get an API Key

The below was tested on `Codex 0.40`, LiteLLM `1.72.2` (api.studio.genai.cba), MacOS `15.5 (24F74)`, zsh `5.9` & bash `3.2.57(1)`

Codex Setup Guide

Installation

Follow the official documentation for the most up-to-date installation instructions: GitHub - openai/codex: Lightweight coding agent that runs in your terminal

Current installation command: `npm install -g @openai/codex` or `brew install codex`

Authentication

Follow the official documentation for the most up-to-date authentication instructions:

1. Update/Save the `sk-` key in the Keychain:

   `security delete-generic-password -s "LITELLM_API_KEY"; security add-generic-password -s "LITELLM_API_KEY" -a "$(whoami)" -w "KEY"`

   where `KEY` is your `sk-` key from Open-WebUI (https://studio.genai.cba)

2. Add the following line to your shell configuration file (`~/.zshrc` , `~/.bashrc` , etc.)

```
export LITELLM_MASTER_KEY=$(security find-generic-password -s "LITELLM_API_KEY" -w)
export OPENAI_API_KEY="$LITELLM_MASTER_KEY"
```

3. Restart your terminal or run `source ~/.zshrc` (or equivalent for your shell)

Configuration

Codex settings

Create or update your `~/.codex/config.toml` file with the following configuration:

I built it manually based on: codex/docs/config.md at main · openai/codex

```
# Recall that in TOML, root keys must be listed before tables.
model = "gpt-5_v2025-08-07_EASTUS2"
model_provider = "litellm"

approval_policy = "on-failure"
sandbox_mode = "danger-full-access" # "workspace-write", or "read-only"
tools.web_search = true
```

```
# Setting `profile` is equivalent to specifying `--profile o3` on the command
# line, though the `--profile` flag can still be used to override this value.
profile = "gpt5"

[model_providers.litellm]
name = "Chat Completions"
base_url = "https://api.studio.genai.cba/v1"
env_key = "OPENAI_API_KEY"
wire_api = "chat"


[profiles.gpt5]
model = "gpt-5_v2025-08-07_EASTUS2"
model_provider = "litellm"
approval_policy = "on-failure"
request_max_retries = 4            # retry failed HTTP requests
stream_max_retries = 10            # retry dropped SSE streams
stream_idle_timeout_ms = 300000    # 5m idle timeout
model_reasoning_effort = "minimal" # minimal, low, medium (default), or high
model_reasoning_summary = "none"   # none, auto, concise or detailed
model_verbosity = "low"            # low, medium or high
sandbox_mode = "danger-full-access" # "workspace-write", or "read-only"
tools.web_search = true
```

Environment Vartiables

MacOS

In our Shell config (e.g. ~/.zshrc), create the following entries:

```
export OPENAI_API_KEY="$LITELLM_MASTER_KEY"           # Already performed in Authentication
export OPENAI_API_BASE="https://api.studio.genai.cba"
export OPENAI_BASE_URL="https://api.studio.genai.cba"
```

⚠️ Alas it seems `export OPENAI_MODEL="gpt-5_v2025-08-07_EASTUS2"` is not recognised.
Also there appear to be a glitch where codex will read `GOOGLE_GEMINI_BASE_URL="http://localhost:9201"` (if declared), so you might have to unset this...

Windows

Same as above using Windows System Properties / Env Var:
See → 🔐Group Internal Root CAs Certificate Store | Method 1: Using System Properties (Recommended)

Token Allocation Strategy

This step was unconclusive since our team did not input figures for max output / token.

The unconclusive tests...

Request

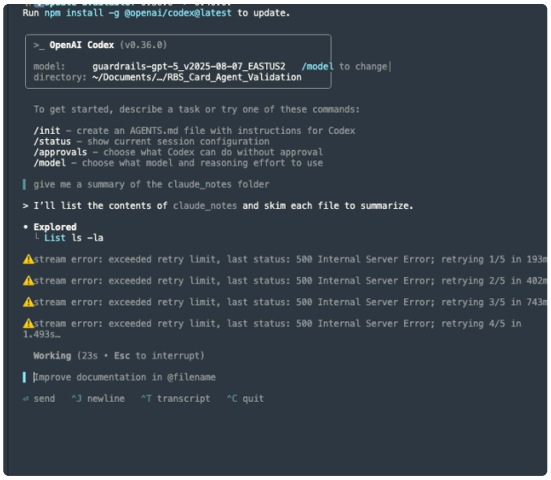
```
curl -X 'GET' \
  'https://api.studio.genai.cba/api/v1/v1/model/info?litellm_model_id=3f705214-f147-406a-9b80-5de2d76662a1' \
  -H 'accept: application/json' \
  -H 'x-litellm-api-key: sk-1234567890'
```

Response

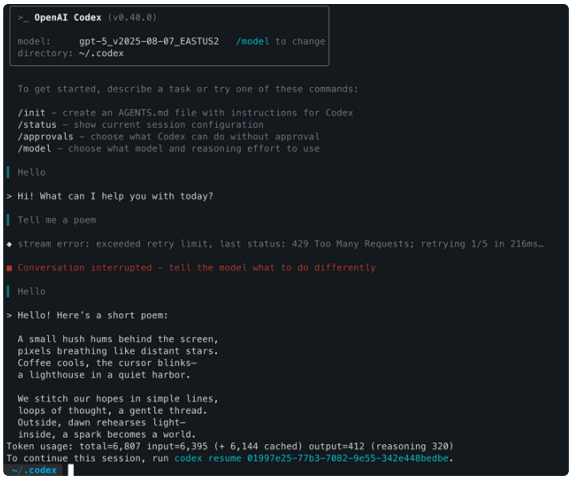

```
{
      "model_name": "guardrails-gpt-5_v2025-08-07_EASTUS2",
      "litellm_params": {
        "api_base": "http://localhost:9099/v1",
        "use_in_pass_through": false,
        "use_litellm_proxy": false,
        "merge_reasoning_content_in_choices": false,
        "model": "openai/guardrails.gpt-5_v2025-08-07_EASTUS2"
      },
      "model_info": {
        "id": "0f74ca6b1ff0bc3b9629c0690897815a5f12213182d476598119b264dfcdcdce",
        "db_model": false,
        "key": "openai/guardrails.gpt-5_v2025-08-07_EASTUS2",
        "max_tokens": null,
        "max_input_tokens": null,
        "max_output_tokens": null,
        "input_cost_per_token": 0,
        "cache_creation_input_token_cost": null,
        "cache_read_input_token_cost": null,
        "input_cost_per_character": null,
        "input_cost_per_token_above_128k_tokens": null,
        "input_cost_per_token_above_200k_tokens": null,
        "input_cost_per_query": null,
        "input_cost_per_second": null,
        "input_cost_per_audio_token": null,
        "input_cost_per_token_batches": null,
        "output_cost_per_token_batches": null,
        "output_cost_per_token": 0,
        "output_cost_per_audio_token": null,
        "output_cost_per_character": null,
        "output_cost_per_reasoning_token": null,
        "output_cost_per_token_above_128k_tokens": null,
        "output_cost_per_character_above_128k_tokens": null,
        "output_cost_per_token_above_200k_tokens": null,
        "output_cost_per_second": null,
        "output_cost_per_image": null,
        "output_vector_size": null,
        "litellm_provider": "openai",
        "mode": null,
        "supports_system_messages": null,
        "supports_response_schema": null,
        "supports_vision": null,
        "supports_function_calling": null,
        "supports_tool_choice": null,
        "supports_assistant_prefill": null,
        "supports_prompt_caching": null,
        "supports_audio_input": null,
```

```
        "supports_audio_output": null,
        "supports_pdf_input": null,
        "supports_embedding_image_input": null,
        "supports_native_streaming": null,
        "supports_web_search": null,
        "supports_url_context": null,
        "supports_reasoning": null,
        "supports_computer_use": null,
        "search_context_cost_per_query": null,
        "tpm": null,
        "rpm": null,
        "supported_openai_params": [
          "frequency_penalty",
          "logit_bias",
          "logprobs",
          "top_logprobs",
          "max_tokens",
          "max_completion_tokens",
          "modalities",
          "prediction",
          "n",
          "presence_penalty",
          "seed",
          "stop",
          "stream",
          "stream_options",
          "temperature",
          "top_p",
          "tools",
          "tool_choice",
          "function_call",
          "functions",
          "max_retries",
          "extra_headers",
          "parallel_tool_calls",
          "audio",
          "web_search_options",
          "response_format"
        ]
      }
```


Regardless, the Codex CLI doesn't expose granular controls for separate input/output caps, so we'll need to manage headroom ourselves against GPT-5's ~400k-token context window and ~128k max output. I'll start with `model_reasoning_effort="minimal"` to reduce internal token burn and latency, then report on "low, medium, high" options that trade cost and latency for quality, especially on long-context runs (where GPT-5 high stays strong at 128k–256k tokens).



`HTTP 500` screenshot
courtesy of @Dom Nasrabadi



`HTTP 429`

Model Compatibility

Testing Methodology

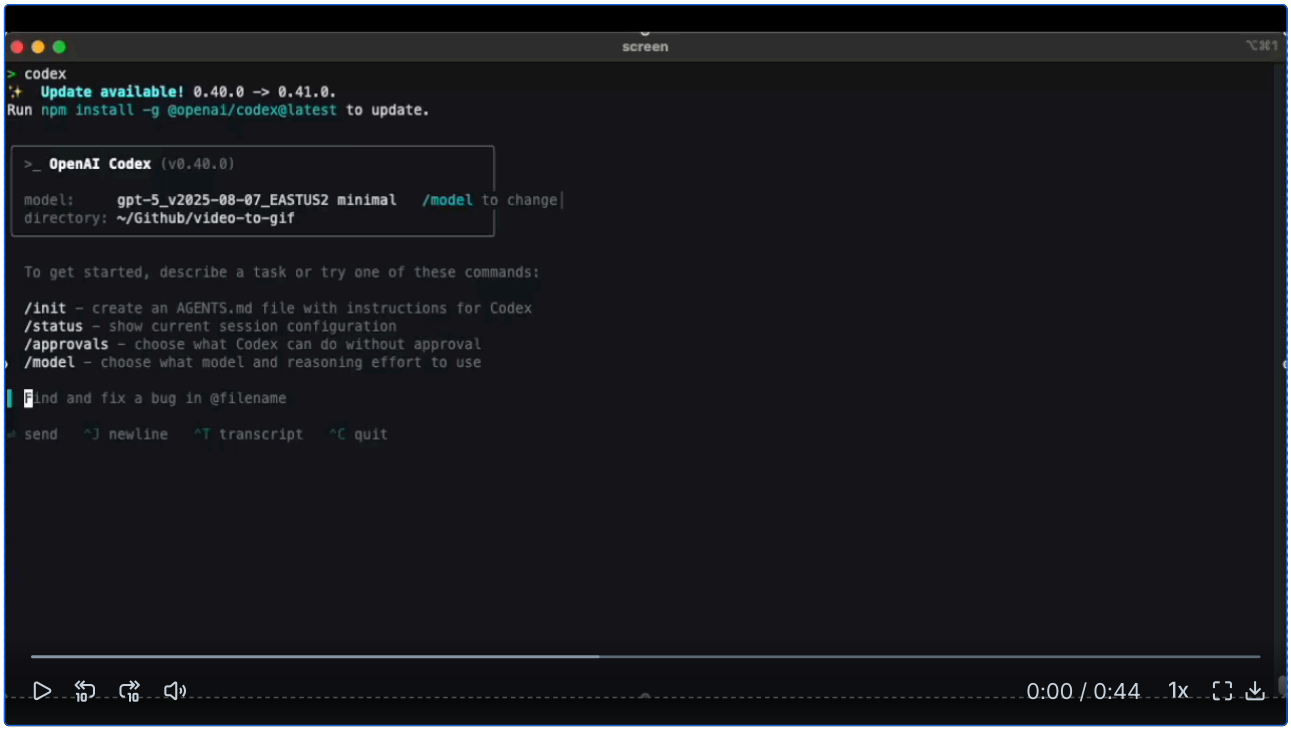
GPT-5

I ran `/init` in a *busy* project directory containing lots of code and pdf/png images. The model performed well and did not crash using the above settings.

Using model_reasoning_effort="minimal" (along with low verbosity and no reasoning summary) reduces hidden “thinking” tokens and extra text, which lowers tokens-per-request and compute time. That can reduce the likelihood of 429s caused by tokens-per-minute quotas and some 500s due to timeouts or overloaded requests. It won’t prevent 429s driven by requests-per-minute limits, nor 500s from provider-side faults. This is require ongoing testing…

Does setting model_reasoning_effort="minimal" on GPT-5 negate its advantages and make it behave like GPT-3/3o?

No, this doesn’t change the underlying model or its capabilities. It only reduces the internal reasoning/compute budget, making responses faster and cheaper but potentially less robust on complex, multi-step tasks. Use minimal for simple tasks; increase effort for harder problems.



codex `/init` and `/status` (sped up)

Other Models WIP

Each model ***will be*** tested using the following procedure:

1. Running `codex --model <model> "What's your model?"` to test if the model responds correctly
2. Recording the response or error message

> 🎉 *Helpful? Drop me a thanks on Achievers ! And if you've got knowledge to share, don't hold back - we all grow when we learn from each other* 💡