Step-by-Step Guide: Running Ollama on Local with Google Colab GPU

This guide walks you through the process of running Ollama models on Google Colab, leveraging Google's cloud infrastructure for increased processing power.

1. Prerequisites

- An **Ollama installation** on your local machine.
- A **Google account**.
- An **Ngrok account** (Sign up for free here).

2. Setting Up Ngrok

1. **Log in to Ngrok**: Access your Ngrok account through their website.
2. **Get Your Authentication Token**:
   - Navigate to the **"Your Authtoken"** section on the left-hand side of the Ngrok interface.
   - Copy the provided token. This token is essential for secure access to your Ngrok account from Google Colab.

3. Setting Up Google Colab

1. **Open the Jupyter Notebook in Google Colab**:
2. **Add Ngrok Authentication Token to Colab**:
   - In Colab, find the "Secrets" section (usually a key icon in the left sidebar).
   - Click **"Add new secret"**.
   - Name the secret **NGROK_AUTH_TOKEN**.
   - Paste your Ngrok authentication token into the "Value" field.
   - Click **"Create secret"**.
3. **Select a Runtime Environment**:
   - At the top-right corner of the Colab screen, ensure you are connected to a runtime.

4. Running the Colab Notebook

```
# Download and run the Ollama Linux install script
!curl -fsSL https://ollama.com/install.sh | sh
```

```
# Get Ngrok authentication token from colab secrets environment
from google.colab import userdata
NGROK_AUTH_TOKEN = userdata.get('NGROK_AUTH_TOKEN')
```

```
# Install:
#  1. aiohttp for concurrent subprocess execution in Jupyter Notebooks
#  2. pyngrok for Ngrok wrapper
!pip install aiohttp pyngrok

import asyncio
import os

# Set LD_LIBRARY_PATH so the system NVIDIA library becomes preferred
# over the built-in library. This is particularly important for
# Google Colab which installs older drivers
os.environ.update({'LD_LIBRARY_PATH': '/usr/lib64-nvidia'})

# Define run - a helper function to run subcommands asynchronously.
# The function takes in 2 arguments:
#  1. command
#  2. environment variable
async def run(cmd):
  print('>>> starting', *cmd)
  p = await asyncio.subprocess.create_subprocess_exec(
      *cmd,
      stdout=asyncio.subprocess.PIPE,
      stderr=asyncio.subprocess.PIPE
  )


# This function is designed to handle large amounts of text data efficiently.
# It asynchronously iterate over lines and print them, stripping and decoding as needed.
  async def pipe(lines):
    async for line in lines:
      print(line.strip().decode('utf-8'))


# Gather the standard output (stdout) and standard error output (stderr) streams of a
subprocess and pipe them through
# the `pipe()` function to print each line after stripping whitespace and decoding UTF-8.
# This allows us to capture and process both the standard output and error messages from
the subprocess concurrently.
  await asyncio.gather(
      pipe(p.stdout),
      pipe(p.stderr),
  )


# Authenticate with Ngrok
await asyncio.gather(
  run(['ngrok', 'config', 'add-authtoken', NGROK_AUTH_TOKEN])
)
# Install:
```

```
#  1. aiohttp for concurrent subprocess execution in Jupyter Notebooks
#  2. pyngrok for Ngrok wrapper
!pip install aiohttp pyngrok

import asyncio
import os

# Set LD_LIBRARY_PATH so the system NVIDIA library becomes preferred
# over the built-in library. This is particularly important for
# Google Colab which installs older drivers
os.environ.update({'LD_LIBRARY_PATH': '/usr/lib64-nvidia'})

# Define run - a helper function to run subcommands asynchronously.
# The function takes in 2 arguments:
#  1. command
#  2. environment variable
async def run(cmd):
  print('>>> starting', *cmd)
  p = await asyncio.subprocess.create_subprocess_exec(
      *cmd,
      stdout=asyncio.subprocess.PIPE,
      stderr=asyncio.subprocess.PIPE
  )


# This function is designed to handle large amounts of text data efficiently.
# It asynchronously iterate over lines and print them, stripping and decoding as needed.
  async def pipe(lines):
    async for line in lines:
      print(line.strip().decode('utf-8'))


# Gather the standard output (stdout) and standard error output (stderr) streams of a
subprocess and pipe them through
# the `pipe()` function to print each line after stripping whitespace and decoding UTF-8.
# This allows us to capture and process both the standard output and error messages from
the subprocess concurrently.
  await asyncio.gather(
      pipe(p.stdout),
      pipe(p.stderr),
  )


# Authenticate with Ngrok
await asyncio.gather(
  run(['ngrok', 'config', 'add-authtoken', NGROK_AUTH_TOKEN])
)
# Run multiple tasks concurrently:
#  1. Start the Ollama server.
#  2. Start ngrok to forward HTTP traffic from the local ollama api running on
localhost:11434.
```

```
#     Instructions come from Ollama doc: https://github.com/ollama/ollama/blob/main/docs/faq.md#how-can-i-use-ollama-with-ngrok
await asyncio.gather(
    run(['ollama', 'serve']),

    # If you don't want to map to a static URL in Ngrok, uncomment line 9 and comment line 10 before running this cell
    run(['ngrok', 'http', '--log', 'stderr', '11434', '--host-header', 'localhost:11434']),
    # run(['ngrok', 'http', '--log', 'stderr, '11434', '--host-header', 'localhost:11434', '--domain', 'insert-your-statik-ngrok-domain-here']),
)
```

at the end the output will have a url to connect local system to colab GPU.

5. Running Ollama on Local : VS Code

```
import os

os.environ["OLLAMA_HOST"] = "url_here"
```

```
!ollama list
```

This guide provides a structured approach to running **Ollama** with **Google Colab's GPU**, making it easier to utilize **high computational power** for model execution.