How-To: Use a DeepInfra Open-Source LLM as a Synthetic Student for the Real-Time Psych Tutor

Target model: meta-llama/Meta-Llama-3.1-8B-Instruct-Turbo (popular open-source family)

Prepared by ChatGPT

September 01, 2025

Overview

This guide shows you how to plug a popular open-source Large Language Model (LLM) hosted on DeepInfra into your real-time psychology tutor as a *synthetic student*. The student bot will fetch questions from the tutor, ask the LLM to pick an answer, then record the results—exactly like a human user—so you can analyze accuracy, response times, mastery, distractor choices, and more.

We will use Meta’s Llama 3.1 family—specifically the 8B Instruct Turbo variant—via DeepInfra’s OpenAI-compatible Chat Completions API. You can swap this for any other DeepInfra model later without changing the harness logic.

Prerequisites

• Python 3.10+ and your tutor server running locally.

• Your OpenAI API key in .env for the tutor’s own generation model.

• A DeepInfra account and API token (we’ll store it as DEEPINFRA_API_KEY in .env).

Your tutor’s generation model remains hard-locked to gpt-5-nano-2025-08-07. The student bot calls DeepInfra separately; no changes are required in the tutor service.

Step 1 — Start the tutor server

Create a virtual environment, install dependencies, and run the server. Then open the web UI at http://localhost:8000 to verify it’s working.

Commands:

```
python3 -m venv .venv
source .venv/bin/activate   # Windows: .venv\Scripts\Activate.ps1
pip install -r requirements.txt
uvicorn server.app:app --reload
```

Health checks (optional):

```
GET /api/skills
POST /api/generate  {"skill_id":"cog-learning-theories",
"type":"mcq", "num_options":5}
POST /api/next      {"user_id":"...", "type":"mcq",
"num_options":5, "verify":true, "use_templates":true}
POST /api/record    {"user_id":"...", "skill_id":"...",
"correct":true, "confidence":3, "time_to_answer_ms":1800}
```

Step 2 — Choose a DeepInfra model

Recommended default:
meta-llama/Meta-Llama-3.1-8B-Instruct-Turbo. It is widely used, fast, and available in DeepInfra’s catalog. You can also try Mistral-7B-Instruct or Mixtral-8x7B-Instruct.

DeepInfra exposes an OpenAI-compatible endpoint at https://api.deepinfra.com/v1/openai. This means you can use the official OpenAI Python client with base_url set to DeepInfra.

Step 3 — Configure credentials

Add your DeepInfra token to the project’s .env file (alongside your existing OpenAI key):

```
DEEPINFRA_API_KEY=di-xxxxxxxxxxxxxxxxxxxxxxxxxx
```

Reload your shell or source the .env so the environment variable is available to Python.

Step 4 — Add the Student Bot harness

Create a file student_bot_deepinfra.py with the code below. It will:
  1) Upsert a user by username.
  2) Request the next MCQ from the tutor.
  3) Ask DeepInfra’s LLM to choose A/B/C/D/E (no explanations).
  4) Record correctness, confidence, and response time back to the tutor.
  5) Repeat for N items.

```
#!/usr/bin/env python3
"""
student_bot_deepinfra.py — synthetic student driven by DeepInfra
Run:
  python student_bot_deepinfra.py --username bot01 --n 300 --model
meta-llama/Meta-Llama-3.1-8B-Instruct-Turbo
"""

import os, time, argparse, random, statistics, requests
from openai import OpenAI

TUTOR_URL = "http://localhost:8000"

def pick_letter_with_llm(stem, options, model):
    # Use DeepInfra's OpenAI-compatible Chat Completions API
    client = OpenAI(
        api_key=os.environ["DEEPINFRA_API_KEY"],
        base_url="https://api.deepinfra.com/v1/openai",
    )
    letters = "ABCDE"[:len(options)]
    prompt = (
        "You are a struggling Intro Psych student. Read the
question and pick ONE answer.\n"
        "Respond with a single character: A, B, C, D, or E. No
explanation.\n\n"
        f"Question: {stem}\n"
        "Options:\n" + "\n".join(f"{letters[i]}. {opt}" for i,opt
in enumerate(options))
    )
    resp = client.chat.completions.create(
        model=model,
        messages=[{"role": "user", "content": prompt}],
        temperature=0.7
    )
    text = resp.choices[0].message.content.strip().upper()
    # Extract first valid letter
    for ch in text:
        if ch in letters:
            return letters.index(ch)
    # fallback random guess
    return random.randrange(len(options))

def record_answer(user_id, skill_id, correct, item_id=None,
confidence=None, t_ms=None):
    body = {"user_id": user_id, "skill_id": skill_id, "correct":
bool(correct)}
    if item_id: body["item_id"] = item_id
    if confidence: body["confidence"] = int(confidence)
    if t_ms: body["time_to_answer_ms"] = int(t_ms)
    r = requests.post(f"{TUTOR_URL}/api/record", json=body,
timeout=30)
    r.raise_for_status()
    return r.json()

def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--username", required=True)
    ap.add_argument("--n", type=int, default=200)
    ap.add_argument("--model", default="meta-llama/Meta-
Llama-3.1-8B-Instruct-Turbo")
    ap.add_argument("--skill", default=None)
    args = ap.parse_args()

    # Upsert user
    r = requests.post(f"{TUTOR_URL}/api/user/upsert",
json={"username": args.username}, timeout=30)
    r.raise_for_status()
    user_id = r.json()["user_id"]

    corrects, latencies = 0, []
    for _ in range(args.n):
        # Request next item
        body = {"user_id": user_id, "type": "mcq", "num_options":
5, "verify": True, "use_templates": True}
        if args.skill: body["current_skill_id"] = args.skill
        nxt = requests.post(f"{TUTOR_URL}/api/next", json=body,
timeout=60).json()

        skill_id = nxt["skill_id"]
        q = nxt["question"]
        stem, options, key = q["stem"], q["options"],
q["correct_index"]

        # Simulate thinking latency (lognormal-ish) and confidence
        think_ms = int(max(600, random.lognormvariate(7.1, 0.35)))
# ~1–20 s
        time.sleep(think_ms / 1000.0)

        pick = pick_letter_with_llm(stem, options,
model=args.model)
        is_correct = (pick == key)
        conf = random.choices([2,3,4], weights=[2,5,3])[0]

        record_answer(user_id, skill_id, is_correct,
item_id=q.get("item_id"), confidence=conf, t_ms=think_ms)
        corrects += int(is_correct)
        latencies.append(think_ms)

    print({
        "user": args.username,
        "n": args.n,
        "acc": round(corrects / max(1, args.n), 3),
        "mean_ms": int(statistics.mean(latencies))
    })

if __name__ == "__main__":
    main()
```

Step 5 — Run it

With the tutor server running, execute the harness. You can run several bots in parallel by launching multiple processes with different usernames.

```
export $(grep -v '^#' .env | xargs)   # ensure DEEPINFRA_API_KEY is
in your env
python student_bot_deepinfra.py --username bot01 --n 500
python student_bot_deepinfra.py --username bot02 --n 500 --model
meta-llama/Meta-Llama-3.1-8B-Instruct
python student_bot_deepinfra.py --username bot03 --n 500 --skill
cog-learning-theories
```

Step 6 — Validate and analyze

As runs complete, open the web UI's Progress tab to see per-category accuracy and per-skill mastery. Programmatically, you can fetch stats via GET /api/user/{user_id}/stats. For item analysis, augment the harness to write a CSV log that includes (user_id, skill_id, item_id, chosen_index, was_correct, confidence, time_ms, timestamp) and compute p-values by item or distractor-wise selection rates.

Behavior controls (make the bot realistic)

• Temperature/top-p: increase to make answers less deterministic.

• Lapse rate: with small probability, guess randomly to mimic slips.

• Position bias: bias slightly toward B/C to reflect naive patterns.

• Speed-accuracy tradeoff: correlate shorter latencies with lower accuracy.

• Domain confusions: inject classical vs. operant conditioning confusions to create meaningful wrong answers.

Notes & Safety

• Never pass correct_index to the LLM. The bot should only see the stem and options.

• Mark synthetic data in any downstream analysis so you can separate it from human runs.

• Respect model and data licenses. Llama 3.1 and Mistral are released under their respective open licenses.

• Keep PII out of logs. The default system stores only username and aggregate stats.

Appendix — Useful Endpoints & Flags

Core tutor endpoints:
  • POST /api/next — get an adaptive MCQ with { skill_id, reason, question }
  • POST /api/record — record { user_id, skill_id, correct, confidence?, time_to_answer_ms? }
  • POST /api/user/upsert — create or retrieve a user by username
  • GET /api/user/{user_id}/stats — snapshot of per_skill and per_category
Generation quality toggles when requesting items:
  • verify: true — self-verify key before serving
  • use_templates: true — use curated, misconception-aligned patterns

Appendix — DeepInfra quick reference

OpenAI-compatible endpoint:
https://api.deepinfra.com/v1/openai
Example (Python, using official openai client):

```
from openai import OpenAI
client = OpenAI(api_key=os.environ['DEEPINFRA_API_KEY'],
base_url='https://api.deepinfra.com/v1/openai')
resp = client.chat.completions.create(
    model='meta-llama/Meta-Llama-3.1-8B-Instruct-Turbo',
    messages=[{'role':'user','content':'Hello'}]
)
print(resp.choices[0].message.content)
```

Other solid choices on DeepInfra:

- mistralai/Mistral-7B-Instruct-v0.2 (small, fast)
- mistralai/Mistral-7B-Instruct-v0.3 (newer tokenizer, function calling)
- mistralai/Mixtral-8x7B-Instruct-v0.1 (MoE, stronger, higher cost)