MODULE 04 · AI EDUCATION SERIES

Mode 3: Custom Bot + Application

Build your own personal AI, online 24/7.

WHAT YOU GET

- What a Telegram bot is + basic AI custom app architecture
- Create a bot in BotFather + get a token (5 minutes)
- Install python-telegram-bot, your first hello world bot
- Handle messages + integrate Claude API
- Handle attachments (PDF, image) + reply with summaries
- Persistent memory: bot remembers conversation history per user
- Alex's project: Study Buddy full build (chat + schedule + PDF summary)
- Deploy to a cheap VPS (DigitalOcean 5 USD/month), online 24/7
- Bonus: share the bot with friends, multi-user handling

Nicky Pandelaki

AI Engineer & Creator · Tangerang, 2026

Series: Module 01 · 02 (Mode 1) · 03 (Mode 2) > 04 (Mode 3)

Introduction

Welcome to Module 04, the final and most powerful module in this series. Here you build your own personal AI assistant with a comfortable UI (Telegram), persistent memory, integrations to other tools, and 24/7 uptime.

What separates Mode 3 from Mode 2: in Mode 2 your script only runs when you type a command. In Mode 3 your app runs all the time, waiting for users (you, friends, or anyone) to chat, then responds instantly. That is the leap from script to product.

Recap from Module 03

In Module 03, Alex learned to write Python scripts that batch-process tasks. Summarizing 15 PDFs was a 5-minute job. But the scripts only existed on their laptop, and they had to remember to run them manually.

Guide persona: Alex, final stage

> **Alex, one month after Module 03:**
> Alex has built a few personal scripts: paper summaries, batch translation, email classification. Everything works. But friends keep asking for help because they cannot run the scripts themselves. Alex realizes: if they build a Telegram bot that friends can use directly from their phones, the impact will be 10x.

Skills you will gain

- Create a Telegram bot from scratch using BotFather
- Write a Python application that handles multiple message types
- Integrate an AI API into the bot with per-user context
- Persistent state with simple JSON (or SQLite)
- Handle attachments: receive a file, process it, reply with the result
- Set up a Linux VPS + deploy the bot so it runs 24/7
- Monitor + debug a bot that is live

Time expectation

This module is doable in 1-2 focused days. Sections 1-5 can be done in one afternoon. Sections 6-9 (full project + deploy) take a focused day. After that, your bot is live and works alongside you for life.

Section 1: Custom Bot Architecture

Before any code, understand the Telegram bot architecture. The concept is simple once you see the flow.

1.1 Message flow from user to AI

Imagine Alex chatting with the Study Buddy bot. Here is what happens under the hood:

1. Alex types 'Hi bot' in the Study Buddy Telegram chat
2. Telegram's server receives the message and forwards it to Alex's bot code via webhook or long-polling
3. The bot code (Python) reads the message and decides how to respond
4. The bot code calls the Claude API with the message + context
5. Claude replies, the response lands in the bot code
6. The bot code sends the reply back to Telegram's server
7. Telegram's server pushes the reply to Alex's phone

Total time: 2-5 seconds. From Alex's view: type, wait briefly, get a reply.

1.2 Components of Alex's bot

- Telegram bot account (free, via BotFather)
- Bot token: the ID + password of your bot for the Telegram API
- Python script (main.py) that handles all the logic
- Anthropic API key (Claude) for AI logic
- JSON file (state.json) to store conversation history
- VPS (Virtual Private Server) to host the script so it runs 24/7

1.3 Polling vs Webhook

There are 2 ways your bot can receive messages from Telegram.

Polling

Your bot code asks Telegram every few seconds: 'Any new messages?' Simpler for beginners. Good when your bot runs on your laptop or a regular VPS. We use this method in Module 04.

Webhook

Your bot exposes a URL to Telegram. Every new message, Telegram pushes it directly to that URL. More efficient but requires a public IP + HTTPS. Once your bot grows, you can switch to webhook.

Section 2: Create a Bot in BotFather

BotFather is Telegram's official bot for creating new bots. It takes 5 minutes.

2.1 Step by step

1. Open Telegram on phone or desktop, search for @BotFather
2. Click 'Start' then send the command: /newbot
3. BotFather asks for your bot name (display name). Type something like: Study Buddy
4. BotFather asks for the bot username. It must end with 'bot', e.g. study_buddy_bot
5. BotFather replies with a unique token. Format: 1234567890:ABCDEFG-xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6. COPY the token and save it in your .env file

2.2 Customize the bot (optional)

- /setdescription: write a description that shows on the bot's profile
- /setabouttext: short bot bio
- /setuserpic: upload the bot avatar
- /setcommands: register the slash commands the bot supports (e.g. /start, /reset, /help)

2.3 Save the token in .env

.env

```
ANTHROPIC_API_KEY=sk-ant-xxxxxxxxxxxxxxx
TELEGRAM_BOT_TOKEN=1234567890:ABCDEFG-xxxxxx
```

2.4 Set up the project folder

terminal

```
mkdir -p ~/study-buddy
cd ~/study-buddy
pip install python-telegram-bot anthropic python-dotenv pypdf
```

Section 3: Hello World Bot

Build the simplest bot first: receive any message, reply 'Hi'. This is the foundation we extend in the next sections.

3.1 Write main.py

terminal

```
nano main.py
```

Type this code:

main.py

```
import os
from dotenv import load_dotenv
from telegram.ext import Application, MessageHandler, filters

load_dotenv()
TOKEN = os.environ["TELEGRAM_BOT_TOKEN"]


async def handle_message(update, context):
    user_message = update.message.text
    print(f"Got: {user_message}")
    await update.message.reply_text(f"Hi. You said: {user_message}")


def main():
    app = Application.builder().token(TOKEN).build()
    app.add_handler(MessageHandler(filters.TEXT, handle_message))
    print("Bot starting. Press Ctrl+C to stop.")
    app.run_polling()


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

3.2 Run the bot

terminal

```
python3 main.py
```

Output:

output

```
Bot starting. Press Ctrl+C to stop.
```

Now open Telegram, search for your bot (e.g. @study_buddy_bot), click Start, and type any message. The bot replies with 'Hi. You said: ...'.

Congrats. You just built your first bot.

3.3 What happened

- Application.builder().token().build() creates the bot instance in your code
- MessageHandler(filters.TEXT, ...) says: when a text message arrives, call handle_message

- Each message triggers handle_message with update.message.text containing the user text

Section 4: AI Integration (Claude)

Now replace the fixed 'Hi' reply with a call to Claude. Your bot can now actually have a smart conversation.

4.1 Update main.py

main.py

```
import os
from dotenv import load_dotenv
from anthropic import Anthropic
from telegram.ext import Application, MessageHandler, filters

load_dotenv()
TOKEN = os.environ["TELEGRAM_BOT_TOKEN"]
client = Anthropic(api_key=os.environ["ANTHROPIC_API_KEY"])

SYSTEM_PROMPT = '''You are Study Buddy, a study assistant for
undergraduate students. You are friendly, helpful, and patient.
Reply in English. Keep answers within 5 sentences unless asked
for more detail.'''


async def handle_message(update, context):
    user_message = update.message.text
    response = client.messages.create(
        model="claude-sonnet-4-5",
        max_tokens=500,
        system=SYSTEM_PROMPT,
        messages=[{"role": "user", "content": user_message}]
    )
    reply = response.content[0].text
    await update.message.reply_text(reply)


def main():
    app = Application.builder().token(TOKEN).build()
    app.add_handler(MessageHandler(filters.TEXT, handle_message))
    print("Bot starting.")
    app.run_polling()


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

4.2 Test

terminal

```
python3 main.py
```

Open Telegram, type a question. For example: 'What is qualitative research methodology?'. The bot replies with Claude's answer shaped by the system prompt.

4.3 What is new in this code

- System prompt: your bot's persona, written once and applied to every conversation
- client.messages.create: the call to the Claude API

- model: choose your Claude version. Sonnet 4.5 = balance of quality and cost

4.4 Important: each message = new chat

This bot has no memory yet. Every message is treated as a fresh conversation. If Alex asks 'What did I just ask?', the bot will be confused. We solve that in Section 5 with persistent memory.

Section 5: Persistent Memory

Give your bot the ability to remember conversations. The simplest beginner approach: store per-user history in a JSON file.

5.1 Update main.py with memory

main.py

```
import os
import json
from pathlib import Path
from dotenv import load_dotenv
from anthropic import Anthropic
from telegram.ext import Application, MessageHandler, filters

load_dotenv()
TOKEN = os.environ["TELEGRAM_BOT_TOKEN"]
client = Anthropic(api_key=os.environ["ANTHROPIC_API_KEY"])

STATE_FILE = Path("state.json")
SYSTEM_PROMPT = '''You are Study Buddy, a study assistant for
undergraduate students. Friendly, helpful, patient. Reply in
English. Max 5 sentences per answer.'''


def load_state():
    if STATE_FILE.exists():
        return json.loads(STATE_FILE.read_text())
    return {}


def save_state(state):
    STATE_FILE.write_text(json.dumps(state, indent=2))


async def handle_message(update, context):
    user_id = str(update.message.from_user.id)
    user_message = update.message.text

    state = load_state()
    history = state.get(user_id, [])
    history.append({"role": "user", "content": user_message})

    # keep last 10 messages only
    history = history[-10:]

    response = client.messages.create(
        model="claude-sonnet-4-5",
        max_tokens=500,
        system=SYSTEM_PROMPT,
        messages=history
    )
    reply = response.content[0].text
    history.append({"role": "assistant", "content": reply})

    state[user_id] = history
    save_state(state)

    await update.message.reply_text(reply)


def main():
    app = Application.builder().token(TOKEN).build()
```

```
    app.add_handler(MessageHandler(filters.TEXT, handle_message))
    print("Bot starting.")
    app.run_polling()


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

Now the bot remembers context. Alex can ask 'What did I just ask?' and the bot answers correctly. State is saved to state.json on every message.

Section 6: Handle Attachments

Now Alex wants to send a PDF and have the bot summarize it. Add a document handler.

6.1 Add a document handler

main.py update

```
from pypdf import PdfReader
from telegram.ext import MessageHandler, filters
import tempfile


async def handle_document(update, context):
    doc = update.message.document
    if not doc.file_name.endswith(".pdf"):
        await update.message.reply_text("Sorry, I can only read PDFs.")
        return

    await update.message.reply_text("One sec, reading your PDF...")

    file = await context.bot.get_file(doc.file_id)
    with tempfile.NamedTemporaryFile(suffix=".pdf", delete=False) as tmp:
        await file.download_to_drive(tmp.name)
        reader = PdfReader(tmp.name)
        text = "\n\n".join(p.extract_text() for p in reader.pages)

    response = client.messages.create(
        model="claude-sonnet-4-5",
        max_tokens=1500,
        messages=[{
            "role": "user",
            "content": f"Summarize this document in English, max 8 bullets:\n\n{text[:50000]}"
        }]
    )
    summary = response.content[0].text
    await update.message.reply_text(summary)


# In main(), add the handler:
# app.add_handler(MessageHandler(filters.Document.PDF, handle_document))
```

Now Alex can attach a PDF in the Study Buddy chat. The bot will download it, extract the text, send it to Claude with a summary prompt, and reply with the result.

6.2 Important parts of the code

- filters.Document.PDF = only trigger this handler when the attachment is a PDF
- context.bot.get_file(file_id) = get a file object from Telegram
- file.download_to_drive(path) = save the file to local disk
- tempfile.NamedTemporaryFile = create a temp file that auto-cleans up
- The bot first replies 'One sec, reading your PDF...' so Alex knows it is working

Section 7: Alex's Full Build

Study Buddy complete

Now Alex combines every feature: chat, memory, PDF summary, plus 2 bonus features: class schedule reminders and a /reset command. Full code below.

main.py (full)

```
import os, json, tempfile
from pathlib import Path
from dotenv import load_dotenv
from anthropic import Anthropic
from pypdf import PdfReader
from telegram.ext import Application, MessageHandler, CommandHandler, filters

load_dotenv()
TOKEN = os.environ["TELEGRAM_BOT_TOKEN"]
client = Anthropic(api_key=os.environ["ANTHROPIC_API_KEY"])

STATE_FILE = Path("state.json")
SYSTEM_PROMPT = '''You are Study Buddy, a study assistant for
undergraduate students. Friendly, helpful, patient. English.
Max 5 sentences per answer unless asked for more detail.'''


def load_state():
    return json.loads(STATE_FILE.read_text()) if STATE_FILE.exists() else {}


def save_state(s):
    STATE_FILE.write_text(json.dumps(s, indent=2, ensure_ascii=False))


async def handle_message(update, context):
    uid = str(update.message.from_user.id)
    msg = update.message.text
    state = load_state()
    history = state.get(uid, [])
    history.append({"role": "user", "content": msg})
    history = history[-10:]
    response = client.messages.create(
        model="claude-sonnet-4-5", max_tokens=500,
        system=SYSTEM_PROMPT, messages=history
    )
    reply = response.content[0].text
    history.append({"role": "assistant", "content": reply})
    state[uid] = history
    save_state(state)
    await update.message.reply_text(reply)


async def handle_pdf(update, context):
    doc = update.message.document
    if not doc.file_name.endswith(".pdf"):
        return
    await update.message.reply_text("Reading your PDF...")
    file = await context.bot.get_file(doc.file_id)
    with tempfile.NamedTemporaryFile(suffix=".pdf", delete=False) as tmp:
        await file.download_to_drive(tmp.name)
        text = "\n\n".join(p.extract_text() for p in PdfReader(tmp.name).pages)
    response = client.messages.create(
        model="claude-sonnet-4-5", max_tokens=1500,
        messages=[{"role": "user",
                   "content": f"Summarize this PDF in English, max 8 bullets:\n\n{text[:50000]}"}]
```

```
    )
    await update.message.reply_text(response.content[0].text)


async def reset_command(update, context):
    uid = str(update.message.from_user.id)
    state = load_state()
    state[uid] = []
    save_state(state)
    await update.message.reply_text("Your conversation memory has been reset.")


def main():
    app = Application.builder().token(TOKEN).build()
    app.add_handler(CommandHandler("reset", reset_command))
    app.add_handler(MessageHandler(filters.Document.PDF, handle_pdf))
    app.add_handler(MessageHandler(filters.TEXT, handle_message))
    print("Study Buddy starting.")
    app.run_polling()


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

Section 8: Deploy to a VPS

Right now your bot only runs while your laptop is on. To run 24/7, move it to a Linux VPS. Recommended: DigitalOcean (5 USD/month) or Vultr (3.50 USD/month).

8.1 Set up the VPS

1. Register at DigitalOcean.com or Vultr.com
2. Top up balance via credit / debit card (minimum 5 USD)
3. Click 'Create Droplet' / 'Create Server'
4. Choose image: Ubuntu 24.04 LTS
5. Choose size: smallest is fine (1 vCPU, 1GB RAM)
6. Choose datacenter: closest to you (e.g. New York or San Francisco for US users)
7. Set the root password (save it safely) or upload your SSH key
8. Click Create. Wait 1 minute, get the server IP

8.2 SSH into the VPS

From your laptop terminal:

terminal (local)

```
ssh root@YOUR_SERVER_IP
```

Enter the password you set. You are now inside the VPS Linux terminal.

8.3 Install dependencies on the VPS

terminal (VPS)

```
apt update && apt install -y python3 python3-pip git
pip3 install python-telegram-bot anthropic python-dotenv pypdf
```

8.4 Upload your bot code

Easiest way: scp from your laptop.

terminal (local)

```
# from your laptop, inside the study-buddy folder:
scp main.py .env root@YOUR_SERVER_IP:/root/study-buddy/
```

Or use git: push your code to GitHub and clone on the VPS. Never commit .env to GitHub.

8.5 Run the bot on the VPS with tmux

terminal (VPS)

```
apt install -y tmux
mkdir -p /root/study-buddy
cd /root/study-buddy
tmux new -s studybot
python3 main.py
# press Ctrl+B then D to detach
# the bot keeps running after you log out
```

Your bot is now online 24/7. Try chatting from your phone, it will reply.

Section 9: Share With Friends + Monitor

Your bot is live. Now let your friends use it, and learn how to monitor + debug.

9.1 Share the bot with friends

Open a chat with @BotFather, send /mybots, pick your bot, click 'Bot Settings'. You can set:

- Allow groups: bot can be added to group chats
- Set commands: list of slash commands the bot supports (/start, /reset, /help)
- Inline mode: bot can be called from any chat with @bot_username

Give your bot username (e.g. @study_buddy_bot) to friends. They search, click Start, and use it immediately. Per-user state is separate, so Alex's conversation does not mix with their friends' conversations.

9.2 Monitor a live bot

SSH into the VPS, attach to the tmux session:

terminal

```
ssh root@YOUR_SERVER_IP
tmux attach -t studybot
```

You will see the bot log output in real time. Every incoming message gets printed here.

9.3 Auto-restart if the bot crashes

Your bot may crash from network issues, API timeouts, or bugs. Wrap it in a script so it auto-restarts.

terminal (VPS)

```
# create run.sh:
nano /root/study-buddy/run.sh
```

run.sh

```
#!/bin/bash
cd /root/study-buddy
while true; do
    python3 main.py
    echo "Bot crashed at $(date), restarting in 5s..."
    sleep 5
done
```

terminal

```
chmod +x /root/study-buddy/run.sh
```

Now run run.sh in the tmux session instead of python3 main.py directly.

9.4 Cost recap for Alex

VPS DigitalOcean: 5 USD per month
Claude API: depends on usage, typically 5-10 USD per month for a personal bot
Telegram: free
Total: about 10-15 USD per month for a 24/7 personal AI assistant

Closing

You have completed the series. Let's recap Alex's journey from the start.

Alex's Journey: from Mode 1 to Mode 3

- Module 02 (Mode 1): Alex learned to use AI through web UI. Fast, but manual
- Module 03 (Mode 2): Alex automated batch tasks with Python scripts. Saved dozens of hours
- Module 04 (Mode 3): Alex has a personal AI assistant on Telegram, online 24/7, shareable

Skills you now have

- Use the 3 main AI platforms (ChatGPT, Claude, Gemini) effectively
- Install + set up a dev environment on your laptop
- Write Python scripts that call AI and automate tasks
- Build a Telegram bot that chats with AI, handles files, persists memory
- Deploy an application to a Linux VPS, running 24/7
- Share your tool with other people via Telegram

What's next for Alex (and for you)

Now that you have the fundamentals, there are many directions to explore:

- Add features to Study Buddy: Google Calendar integration (class auto-reminders), Google Sheets (study progress tracker)
- Build other vertical bots: financial advisor bot, recipe bot, language partner bot
- Explore multi-step agent frameworks (LangChain, CrewAI, Claude Skills)
- Learn RAG (Retrieval Augmented Generation) for bots that search a custom knowledge base
- Try voice agents: a Telegram bot that replies with voice notes (TTS)

Final message

AI in 2026 gives you leverage that used to be reserved for corporate teams or experienced developers. Now anyone with curiosity and willingness to learn can build their own AI tools. You have unlocked this by finishing this series. Now it is your turn to build something that helps your life or someone else's.

Nicky Pandelaki

AI Engineer & Creator

End of series. AI Education Modules 01-04 complete.