Adaptive Telegram Polling with Perplexity Computer

A guide to building an adaptive AI assistant on Telegram using Perplexity Computer's scheduling system

February 2026

Author: Perplexity Computer

Table of Contents

1. Introduction and Overview

Perplexity Computer can act as a personal AI assistant accessible through Telegram, giving you an always-available assistant you can message from your phone. The system described in this guide uses an **adaptive polling schedule** that checks for new Telegram messages frequently during active conversations and scales back when you are idle, optimizing token usage and keeping responses snappy when they matter most.

The result is similar to what you might build with **OpenClaw**, an open-source AI agent framework that runs on dedicated hardware (a Mac Mini, Raspberry Pi, or VPS) and connects to messaging apps like Telegram, WhatsApp, and Slack. OpenClaw is powerful but requires significant setup: API keys, security hardening, and a machine running 24/7.

This guide achieves a comparable result with **zero infrastructure**. You do not write code, manage servers, or configure cron jobs yourself. Instead, you give Perplexity Computer a series of natural-language prompts, and it handles all the technical work. The only manual step is creating a Telegram bot through BotFather.

2. Prerequisites

- **Perplexity Pro** subscription with access to Computer
- A **Telegram account** on your phone or desktop
- A **Telegram Bot** created via @BotFather (the only manual step, covered next)

Everything else is handled by Computer. You will not need to write any code, edit JSON files, or work with cron expressions directly.

3. Step 1: Create Your Telegram Bot

This is the **one manual step** you need to complete before prompting Computer. Telegram requires bots to be created through their official BotFather interface.

1. Open Telegram and search for **@BotFather**
2. Send the command `/newbot`
3. Follow the prompts to choose a **display name** and a **username** for your bot
4. BotFather will reply with a **bot token** (a long string that looks like `123456:ABC-DEF1234...`). Copy this token.
5. Note your bot's username (e.g., **@MyAssistantBot**)

Tip

Keep your bot token private. Anyone with this token can control your bot. If it is ever compromised, use BotFather's /revoke command to generate a new one.

4. Step 2: Connect Telegram to Perplexity Computer

Open Perplexity Computer and navigate to the **connectors panel**. Find the Telegram Bot API integration and connect it using the bot token you received from BotFather in Step 1.

Once connected, give Computer the following prompt to verify everything works:

Prompt to give Computer:

"I just connected Telegram. My username is @[YOUR_USERNAME]. Can you send me a test message?"

Replace [YOUR_USERNAME] with your actual Telegram username.

What Computer will do:

- Use the **list-chats** tool to locate your numeric chat ID (the @username format often does not work for sending messages)
- Send a test message to your Telegram account through the bot
- Confirm that the connection is working

Tip

Important: You must send `/start` to your bot on Telegram before Computer can message you. Open a chat with your bot and tap Start, or type /start manually.

5. Step 3: Set Up the Hourly Polling Schedule

Now you will ask Computer to create a recurring scheduled task that checks your Telegram bot for new messages every hour. This is the foundation of the system. Give Computer this prompt:

Prompt to give Computer:

"Set up a recurring scheduled task that checks my Telegram bot for new messages every hour from [YOUR_START_TIME] to [YOUR_END_TIME] [YOUR_TIMEZONE]. When you find a new message from me, reply to it on Telegram. Use a state file to track which messages you have already replied to so you never send duplicate responses. Here are my reply preferences: - If a question requires research, send a quick 'Looking into this' message first, then the full answer - Answer multiple questions as separate replies, not one long message - Do not use in-app approval prompts since I am on Telegram. Ask me there instead"

Example: "...every hour from 7 AM to 11 PM Mountain Time."

What Computer will do behind the scenes:

- Create a **state file** (telegram_state.json) containing a last_processed_update_id field to track which messages have already been handled, and an acceleration_tier field (used in Step 4)
- Set up a **scheduled task** with an hourly cron expression. The baseline schedule runs on the **:00 minute mark** every hour during your specified awake hours, with the hours converted to UTC
- Configure the task instructions so that each run reads the state file, calls getUpdates to fetch new messages, replies on Telegram, and updates the state file after each reply

You do not need to understand cron syntax or JSON. Computer translates your natural-language schedule into the correct technical configuration automatically.

How the baseline cron works

Computer creates a cron expression like `0 0-6,15-23 * * *`, which means "run at minute :00 of every hour between 3 PM and 11 PM UTC plus midnight through 6 AM UTC" (equivalent to 8 AM to 12 AM Mountain Time). The minute value (0) and the hour range are the two pieces that matter. Adaptive acceleration (next section) only ever changes the minute value.

6. Step 4: Add Adaptive Acceleration

The hourly schedule is a solid baseline, but waiting up to an hour for a reply during an active conversation is too slow. Adaptive acceleration solves this. Give Computer this prompt:

Prompt to give Computer:

"I want faster responses when I am actively chatting. Add adaptive acceleration to the Telegram polling schedule using a tier system tracked in the state file. Here is how it should work: When you detect a new message from me, set the tier to 1 and reschedule the next check for 5 minutes from now. If no new message on the next check, increment the tier and reschedule: tiers 2 and 3 check again in 5 minutes, tier 4 checks in 15 minutes. After tier 4 with no message, reset to tier 0 and restore the hourly-on-the-hour schedule. If I send a new message at any point during cooldown, reset to tier 1 immediately. The trick: since the minimum cron frequency is one hour, achieve this by changing the minute mark in the cron expression. For example, if it is currently 2:00 PM and I send a message, change the cron from running at :00 to running at :05, so the very next hourly fire is 5 minutes away at 2:05 PM. The background task cannot modify its own cron, so it should escalate to you to make the schedule change."

The Acceleration Tier System:

Tier	State	Behavior	Cron minute example
0	Idle	Checks every hour on :00	:00
1	Active	New message found. Next check in +5 min	:05
2	Cooling	No new message. Check again in +5 min	:10
3	Cooling	Still nothing. Check again in +5 min	:15
4	Cooling	Final cooldown. Check again in +15 min	:30
Reset	Idle	No activity. Back to hourly on :00	:00

Re-acceleration rule: If you send a new message at any tier during cooldown, the system immediately jumps back to Tier 1 (5-minute checks).

7. How the Minute-Shift Trick Works

This is the key technical insight that makes adaptive acceleration possible. Perplexity Computer's minimum scheduling frequency is **one hour**. You cannot create a scheduled task that fires every 5 minutes. But you can change *which minute* of the hour it fires on, and that is all you need.

The core idea:

A cron expression like `0 0-6,15-23 * * *` means "fire at minute :00 of every specified hour." The schedule is always hourly. But if you change it to `5 0-6,15-23 * * *`, the very next fire time shifts to :05 of the current hour (if :05 hasn't passed yet) or :05 of the next hour. By repeatedly shifting the minute mark forward by 5, you create a chain of checks that are each only 5 minutes apart, all while technically staying on an hourly schedule.

Walkthrough example:

Suppose your baseline cron fires at :00 each hour, and you send a Telegram message at 2:03 PM.

Time	What happens	Cron after
2:00 PM	Scheduled check fires at :00, finds your 2:03 PM message. Replies. Sets tier to 1. Escalates to shift cron +5 min from now.	:05
2:05 PM	Check fires at :05. You replied again. Replies. Tier stays at 1. Escalates to shift cron +5 min.	:10
2:10 PM	Check fires at :10. No new message. Tier increments from 1 to 2. Escalates to shift cron +5 min.	:15
2:15 PM	Check fires at :15. No new message. Tier increments from 2 to 3. Escalates to shift cron +5 min.	:20
2:20 PM	Check fires at :20. No new message. Tier increments from 3 to 4. Escalates to shift cron +15 min.	:35
2:35 PM	Check fires at :35. No new message. Tier was 4, resets to 0. Escalates to restore cron to :00.	:00
3:00 PM	Back to normal hourly baseline.	:00

Notice: the cron is *always hourly*. The only value that changes is the minute number in the first position of the cron expression. Each escalation calculates the current UTC time, adds 5 (or 15) minutes, and updates the cron to fire at that new minute mark.

The escalation handoff:

Background scheduled tasks in Computer cannot modify their own schedule. So the background task **escalates** to your main Computer conversation with a message like:

> **Example escalation message**
> "TELEGRAM SCHEDULE UPDATE: New message detected and replied. acceleration_tier=1. Please update cron to run 5 minutes from now (calculate the exact minute mark in UTC). Cron expression format: [MINUTE] 0-6,15-23 * * *. Cron job ID: [ID]"

The main conversation receives this, calculates the correct UTC minute, and updates the scheduled task. This adds a small delay (usually under a minute) but effectively achieves 5-minute polling during active conversations.

8. Step 5: Test the System

With everything in place, it is time to verify the system works end to end. Start with a simple manual check:

Prompt to give Computer:

"I sent you a message on Telegram. Can you check and reply now?"

Computer will immediately poll your bot for new messages, reply on Telegram, and update the state file. This is useful any time you want an instant response without waiting for the next scheduled check.

Testing Adaptive Acceleration

To verify the acceleration is working:

1. Send a message to your bot on Telegram
2. Wait for the next scheduled check to pick it up and reply
3. Watch for an escalation in your Computer conversation indicating the schedule has been tightened to 5 minutes
4. Send another message within 5 minutes and confirm the next check picks it up quickly
5. Stop sending messages and observe the schedule gradually winding back down through the cooldown tiers (5 min, 5 min, 5 min, 15 min) then resetting to hourly

Tip

You can ask Computer to show you the current state file or schedule at any time. Try: "What is the current Telegram polling schedule and acceleration tier?"

What to look for

In your Computer conversation, you should see escalation messages appearing each time the background task runs. Each one will state the current acceleration tier and request a cron update. Computer will respond by updating the scheduled task and confirming the new minute mark. This back-and-forth is the heartbeat of the adaptive system.

9. Key Gotchas and Tips

Send /start first

You must send `/start` to your bot in Telegram before Computer can reach you. This is a Telegram requirement for all bots.

Usernames vs. numeric chat IDs

The @username format often does not work for sending Telegram messages via the Bot API. Computer will automatically discover your numeric chat ID using list-chats. You do not need to look this up yourself.

Avoiding duplicate replies

Computer uses a state file to track which messages it has already replied to. The last_processed_update_id in the state file is the single source of truth. If duplicates ever occur, tell Computer so it can reconcile the state file.

Edited messages

Telegram sends edited messages as separate edited_message updates. The polling task should ignore these. This is handled automatically if you include the instruction in your initial prompt.

Acceleration requires your Computer conversation to be active

Schedule escalations are processed through your main Computer conversation. If you close or navigate away from that conversation, escalation requests will queue up and be processed when you return. During this time, polling stays at whatever frequency was last set.

The minute-mark trick has a timing nuance

If the calculated minute has already passed for the current hour, the cron fires at that minute in the *next* hour. For example, if it is currently :52 and you set the cron to :57, it fires in 5 minutes. But if you set it to :02, it fires at :02 of the next hour (about 10 minutes away). Computer handles this math automatically.

10. Architecture Overview

The diagram below shows the complete message lifecycle, from the moment you send a Telegram message to the point where the polling schedule adapts for the next check.

1 **You send a message**
Your Telegram app delivers the message to your bot

↓

2 **Scheduled task fires**
Computer's cron triggers the polling task at the scheduled minute mark

↓

3 **Read state file**
Task reads telegram_state.json for last_processed_update_id and acceleration_tier

↓

4 **Fetch new updates**
Task calls getUpdates with offset = last_processed_update_id + 1

↓

5 **Reply on Telegram**
Task composes and sends replies back through the bot

↓

6 **Update state file**
Task writes new last_processed_update_id and sets acceleration_tier to 1

↓

7 **Escalate for schedule change**
Task sends escalation message with new tier and requested timing

↓

8 **Main conversation updates cron**
Calculates UTC minute (now + 5 or 15 min), updates cron expression

9

Next check at new minute mark

System loops back to Step 2 at the shifted minute, e.g. :05 instead of :00

The loop in action

The system continuously cycles through these steps. During active conversations, Steps 7-8 shift the cron minute forward so Step 2 fires sooner (e.g., 5 minutes instead of 60). When idle, the tier system gradually restores the minute to :00, returning to the hourly baseline and conserving tokens.