

both **chromium-driver** and **chromium-headless-shell** are **designed to be controlled programmatically**, which includes being operated by an AI agent (with permission), via scripts, command-line tools, or automation frameworks. Here's how that works:

chromium-driver (WebDriver)

- **Used with:** Tools like **Selenium**, **Playwright**, or **Puppeteer**.
- **Control method:**
 - You (or an AI agent) write scripts (usually in Python, JavaScript, etc.) to launch and control a real browser session.
 - The WebDriver protocol acts like a remote control for the browser.
- **Capabilities:**
 - Visit websites
 - Click buttons
 - Fill forms
 - Take screenshots
 - Scrape data

✅ **Ideal for:** Full browser automation with UI rendering — for tasks that require interaction with JavaScript-heavy pages.

chromium-headless-shell

- **Used with:** Command-line scripts or frameworks like **Puppeteer**, **Playwright**, or custom headless setups.
- **Control method:**
 - Run Chromium in **headless mode** (no GUI).
 - Useful for generating PDFs, screenshots, crawling sites, etc.
- **Can be controlled via:**
 - Command-line options (e.g., `--headless --screenshot`)
 - Scripts (e.g., via Puppeteer)

✅ **Ideal for:** Background tasks where no visible browser is needed — faster and more efficient for data extraction or testing.

✓ Can an AI Agent Use Them?

Yes, **if permitted**, an AI agent (or automation system) can use these in two main ways:

1. Via Shell or Terminal:

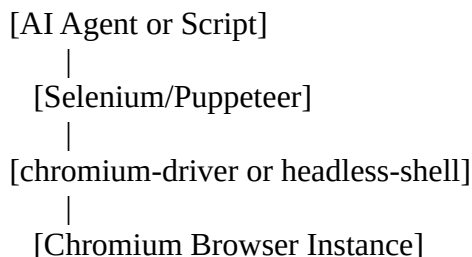
- Directly run Chromium in headless mode.
- Launch `chromedriver` and control via WebDriver clients.

2. Via Automation Libraries:

- Use Python/Node.js scripts to control browser behavior using `chromium-driver` or `headless-shell`.

🧠 Example: AI-Controlled Scraping Bot

A basic architecture might look like this:



here's a **set of LLM training examples** formatted to show how an AI model can **learn to control Chromium via `chromium-driver` or `chromium-headless-shell`**, using realistic shell commands and scripting patterns. These examples are framed as **instruction** → **input** → **output** pairs, which are commonly used in training datasets for instruction-tuned LLMs.

🔧 Training Examples: LLM Shell + Browser Automation via Chromium

EXAMPLE 1 — Use Headless Chromium to Take a Screenshot

- instruction: Take a screenshot of example.com using headless Chromium.

input: |

`chromium --headless --disable-gpu --screenshot=example.png https://example.com`

output: Screenshot saved to example.png

EXAMPLE 2 — Use Chromium to Save a Webpage as PDF

- instruction: Save the OpenAI homepage as a PDF file using Chromium in headless mode.

input: |

```
chromium --headless --disable-gpu --print-to-pdf=openai.pdf https://openai.com
```

output: PDF saved as openai.pdf

EXAMPLE 3 — Launch Selenium with chromium-driver in Python

- instruction: Write a Python script using Selenium to open a page and print its title using Chromium.

input: |

```
from selenium import webdriver
```

```
from selenium.webdriver.chrome.service import Service
```

```
from selenium.webdriver.chrome.options import Options
```

```
options = Options()
```

```
options.add_argument("--headless")
```

```
options.add_argument("--disable-gpu")
```

```
service = Service("/usr/bin/chromedriver") # Adjust path if needed
```

```
driver = webdriver.Chrome(service=service, options=options)
```

```
driver.get("https://example.com")
```

```
print(driver.title)
```

```
driver.quit()
```

output: |

Example Domain

EXAMPLE 4 — Use Puppeteer to Generate Screenshot (Node.js)

- instruction: Use Puppeteer (Node.js) to generate a screenshot of <https://example.com>

input: |

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch({ headless: true });
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({ path: 'example.png' });
  await browser.close();
})();
```

output: Screenshot saved to example.png

EXAMPLE 5 — Use Shell to Check If chromium-driver is Installed

- instruction: Check if `chromedriver` is installed on the system and output the version.

input: |

```
chromedriver --version
```

output: |

```
ChromeDriver 139.0.7258.154
```

EXAMPLE 6 — Command to Launch Chromium with Remote Debugging

- instruction: Start Chromium with remote debugging on port 9222.

input: |

```
chromium --remote-debugging-port=9222 --headless --disable-gpu
```

output: Chromium started in headless mode on port 9222

EXAMPLE 7 — Use `curl` and Headless Chromium to Verify Page Load Time

- instruction: Launch a page in headless Chromium and measure load performance using DevTools.

input: |

```
chromium --headless --disable-gpu --dump-dom https://example.com
```

output: |

```
<!doctype html>
```

```
<html>
```

These examples teach an LLM:

- How to **call Chromium** in headless mode with CLI flags
- How to use `chromium-driver` via **Selenium or Puppeteer**
- How to interpret CLI inputs/outputs for task automation
- How to combine shell scripting with browser control