### Innovate with Intelligence: Build the Future of the Web with Gemini Nano and Chrome AI

**Introduction:**

**AI-powered Chrome Extension** concept called **Magictool AI**, leveraging ChatGPT (or similar LLMs) inside Chrome.

submit to a hackathon like the **Google Chrome Built-in AI Challenge 2025**:

# **Magictool AI — Chrome Extension Powered by ChatGPT**

### **📌 Elevator Pitch**

Magictool AI is your **all-in-one productivity companion** inside Chrome. It brings ChatGPT directly into the browser, contextually aware of the web page you’re viewing. With one click or shortcut, you can **summarize, explain, translate, generate content, or automate actions** — all without leaving your tab.

## **🎯 Key Features**

1. **Page Summarizer**
   * One-click TL;DR of any webpage, article, or PDF.
   * Gemini Nano (on-device) for instant local summaries; ChatGPT API fallback for richer insights.
2. **Smart Q&A Overlay**
   * Highlight text → right-click → “Ask Magictool AI” for contextual explanations or deeper research.
3. **Multi-mode Input**
   * Accepts voice commands (speech-to-text).
   * Can analyze images/screenshots from the page.
4. **Content Creation Tools**
   * Rewrite/translate selected text.
   * Draft emails, social posts, or blog summaries directly from browser content.
5. **Productivity Integrations**
   * Export keynotes into Google Docs/Notion.
   * Auto-generate calendar reminders, tasks, or structured JSON data.
6. **Privacy Mode**
   * Local Gemini Nano summarization → no server calls.
   * Explicit toggle to use ChatGPT API (when needed for deep knowledge).

## **⚙️ Tech Stack**

* **Frontend:** Chrome Extension (Manifest V3, React-based popup UI).
* **AI Backend:**
  + **Gemini Nano via Chrome built-in Prompt API** (fast, private, offline).
  + **ChatGPT API** (OpenAI GPT-4/GPT-5 models) for advanced reasoning.
* **Storage:** IndexedDB (local summaries & preferences).
* **Optional Cloud:** Node.js/Express backend for API proxying and premium features.

## **🖼️ Architecture Flow**

1. User highlights text or clicks Magictool button.
2. Extension grabs page content (DOM, text, optional screenshot/audio).
3. Passes to AI engine:  
   * **Local Gemini Nano (Chrome API)** → quick summarization/extract.
   * **Remote ChatGPT API** (if deeper context requested).
4. Results rendered in popup overlay (summary, rewrite, actions).
5. User can export, copy, or run quick actions.

## **📅 4-Week Build Timeline**

**Week 1:** Extension skeleton, UI popup, text extraction.  
 **Week 2:** Integrate Gemini Nano Prompt API + basic summarizer.  
 **Week 3:** Add ChatGPT API fallback + multimodal input (voice, image).  
 **Week 4:** Polish UX (overlay, shortcuts, right-click menu), testing, and release.

## **✨ Differentiators**

* Hybrid AI: **on-device (Gemini Nano) + cloud (ChatGPT)** → balance speed, cost, and intelligence.
* Contextual browser integration (right-click + overlay).
* Privacy-first (local processing option).
* Multi-modal (text, voice, images).

## **📊 Potential Use Cases**

* Students: summarize research papers.
* Professionals: generate email replies from web content.
* Analysts: extract structured data (budgets, KPIs) from reports.
* Content creators: repurpose articles into posts.

## **🔑 Next Steps**

* Build **MVP** (summarizer + contextual Q&A).
* Add **premium tier** (cloud ChatGPT for advanced queries).
* Explore **enterprise version** (knowledge base + team sharing).

👉 Would you like me to **create the actual Chrome Extension starter code (Manifest V3 + React popup + basic content script)** for Magictool AI, so you can start building right away?

//

**Google Chrome Built-in AI Challenge 2025**: idea + pitch + how it uses **Gemini Nano / Chrome built-in AI APIs** + architecture, milestones, deliverables, demo plan, success metrics and sample code sketch. I built this to target the Devpost/hackathon rules and prize categories and to produce a demo you can actually finish in 3–4 weeks. (I pulled the official challenge, rules and API docs for accuracy.) [Chrome for Developers+2Google Chrome Built-in AI Challenge 2025+2](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)

# **1) Elevator pitch (copy for Devpost)**

**Project name:** InsightBridge — Instant Contextual Summaries & Actions for Any Web Page  
 **One-liner:** Use Gemini Nano inside Chrome to turn any webpage into an interactive, multimodal assistant that summarizes content, extracts action items, visualizes key metrics, and lets users trigger safe automations — all client-side and privacy-friendly.  
 **Why it’s compelling:** InsightBridge demonstrates a real, widely-useful hybrid AI web experience — fast on-device summarization with optional server enrichments (search/knowledge) — and showcases Chrome built-in AI APIs (Prompt API, Summarization, multimodal input) to win the *Most Innovative* or *Best Multimodal* categories. [Google Chrome Built-in AI Challenge+1](https://googlechromeai.devpost.com/?utm_source=chatgpt.com)

# **2) How it uses Chrome built-in AI (short)**

* **Gemini Nano (on-device)** for fast summarization, classification and suggested actions (no network roundtrip for the core UX). [AI Engineer Guide](https://aiengineerguide.com/blog/gemini-nano-chrome/?utm_source=chatgpt.com)
* **Prompt API (Chrome built-in)** with multimodal support to accept page text + screenshot (image) + user voice note, then return structured JSON: {summary, highlights[], actions[]}. [Chrome for Developers](https://developer.chrome.com/docs/ai?utm_source=chatgpt.com)
* **Summarization API** to distill long pages into TL;DR + bullet action items. [Chrome for Developers](https://developer.chrome.com/docs/ai?utm_source=chatgpt.com)

(These are the exact APIs the challenge asks you to use — handy for scoring on “used built-in AI APIs”.) [Chrome for Developers+1](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)

# **3) Key features (MVP → Extra)**

**MVP**

1. Page summarizer: single click → 3-sentence TL;DR + 5 bullet highlights.
2. Action items extractor: identify tasks (e.g., “book demo”, “add to calendar”, “download report”).
3. Multimodal input: accept a screenshot or short voice note to focus the summary.
4. “Quick Actions” palette: one-click options — copy summary, create calendar event, compose email draft, save to notes.
5. Privacy mode: all core inference on device (Gemini Nano) — user data stays local.

**Extras (stretch / prize boosters)**

* LLM-generated visuals: automatically generate a one-slide visual summary (chart or visual card) from page stats.
* Browser extension + progressive web app (PWA) combo.
* Smart multi-tab summary (compare two product pages).
* Exportable JSON output for judges to evaluate structured outputs.

# **4) Why this fits challenge categories & scoring**

* Uses **Gemini Nano** + Chrome built-in AI APIs (rules require that). [Chrome for Developers+1](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)
* **Multimodal** (text + image + audio) — aligns with “Best multimodal AI app” prize. [Google Chrome Built-in AI Challenge 2025](https://googlechromeai2025.devpost.com/rules?utm_source=chatgpt.com)
* Demonstrates practical value / UX — appeals to “Most Innovative” and user-impact criteria. [Google Chrome Built-in AI Challenge 2025](https://googlechromeai2025.devpost.com/rules?utm_source=chatgpt.com)

# **5) Tech stack & components**

* Frontend: **Chrome Extension** (manifest v3) + small PWA React app for UI.
* Built-in AI: **Chrome Prompt API / Summarization** (Gemini Nano on device). [Chrome for Developers](https://developer.chrome.com/docs/ai?utm_source=chatgpt.com)
* Server (optional, for enrichments only): lightweight Node/Express or Firebase — *only* for non-sensitive optional features (shareable summaries, analytics). Keep core inference local for privacy and speed.
* Storage: IndexedDB (local) for saved summaries; server DB optional (Mongo/Postgres).
* CI: GitHub repo + simple GH Pages demo + recorded walkthrough.

# **6) Minimal architecture (diagram text)**

User clicks extension → Extension collects page DOM and (optional) screenshot/audio → Calls Chrome Prompt/Summarization API (client side) → Gemini Nano returns structured response → UX renders summary + suggested actions → Optional server for share/sync.

(Everything critical runs in the browser; server is only for extra features or persistence.)

# **7) Demo & submission deliverables (what judges expect)**

* Public GitHub repo with code + README (how to run locally & test).
* Short video (2–3 minutes) showing core flows: summarization, multimodal input, quick actions. (Check Devpost length rules; some threads show concerns about video length — keep ~3 minutes). [Google Groups](https://groups.google.com/a/chromium.org/g/chrome-ai-dev-preview-discuss/c/WEI7wkvAKqo?utm_source=chatgpt.com)
* Live extension build (packaged or instructions to load unpacked) + PWA demo link (if possible).
* Devpost submission page with screenshots, problem statement, and tech used. [Google Chrome Built-in AI Challenge+1](https://googlechromeai.devpost.com/?utm_source=chatgpt.com)

# **8) 4-week sprint timeline (detailed milestones — finishable & judge-ready)**

### **Week 0 (Prep — 2 days)**

* Register for challenge / read rules and prize categories. [Chrome for Developers+1](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)
* Prepare repo, Chrome extension skeleton, and design wireframes.

**Deliverable:** Repo skeleton + wireframes + short plan.

### **Week 1 (Core inference + UI)**

* Implement page text extraction & send to Prompt/Summarization API.
* Build basic UI panel showing TL;DR + bullets.
* Local IndexedDB save/load.

**Deliverable:** Working local summarizer demo (text only).

### **Week 2 (Multimodal + Actions)**

* Add screenshot capture and voice input (Web Audio API).
* Create Prompt templates to accept image / audio context.
* Build Quick Actions: copy, calendar draft, email template.

**Deliverable:** Multimodal inputs working + action palette.

### **Week 3 (Polish + Privacy + Server opt)**

* Add privacy toggle (local only).
* Improve UX, add animations, edge case handling.
* Optional: minimal server for shareable links (if time).

**Deliverable:** Stable extension, demo flows recorded.

### **Week 4 (Testing, Video, Submission)**

* Test on multiple pages, fix bugs.
* Record 2–3 minute demo video & assemble Devpost assets (description, screenshots).
* Submit to Devpost + Chrome challenge page (follow rules). [Google Chrome Built-in AI Challenge 2025+1](https://googlechromeai2025.devpost.com/?utm_source=chatgpt.com)

# **9) Sample Prompt API usage (conceptual snippet)**

Below is a short conceptual JS sample for how the extension could call the built-in Prompt/Summarization API from the extension page context. **Use the official Chrome docs & samples when implementing**; this is a high-level template to shape your code.

// conceptual — follow Chrome docs for exact API names + auth

async function summarizePage(pageText, imageBlob=null, audioBlob=null) {

const prompt = [

{ role: "system", content: "You are a concise summarizer. Output JSON." },

{ role: "user", content: `Summarize this page and return JSON: ${pageText.slice(0,2000)}` }

];

// if multimodal, attach image/audio per Chrome Prompt API spec

const response = await chrome.ai.prompt({

model: 'gemini-nano',

prompt,

multimodal: {

image: imageBlob, // conceptual

audio: audioBlob // conceptual

},

maxOutputTokens: 400

});

// response.parsedText assumed — parse JSON safe

const parsed = JSON.parse(response.outputText || "{}");

return parsed; // { summary, highlights, actions }

}

⚠️ This is a conceptual example. Use the exact call syntax and security model from Chrome’s developer docs. [Chrome for Developers](https://developer.chrome.com/docs/ai?utm_source=chatgpt.com)

# **10) Judging tips — how to impress judges**

* **Multimodality:** Show audio + image + text working together. (This wins the “Best multimodal” category.) [Google Chrome Built-in AI Challenge 2025](https://googlechromeai2025.devpost.com/rules?utm_source=chatgpt.com)
* **Privacy angle:** Emphasize on-device inference with Gemini Nano — judges love privacy-first demos. [AI Engineer Guide](https://aiengineerguide.com/blog/gemini-nano-chrome/?utm_source=chatgpt.com)
* **Structured outputs:** Return JSON with actions so judges can evaluate correctness programmatically.
* **Polished UX + short video:** Keep the demo crisp (2–3 min) and show a compelling real-world scenario (e.g., summarizing a long product comparison and auto-creating an email to stakeholder).

# **11) Risks & mitigations**

* **API/feature availability:** Chrome built-in APIs may be available in certain Chrome channels (beta/dev). Mitigate by providing a clear README on which Chrome version/channel to use for testing. [Chrome for Developers+1](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)
* **Model limitations:** Gemini Nano is optimized for summarization & classification (not deep factual retrieval). Use it for UX tasks (summarize/extract actions) and avoid positioning it as a factual oracle. [AI Engineer Guide](https://aiengineerguide.com/blog/gemini-nano-chrome/?utm_source=chatgpt.com)

# **12) Submission checklist (copy for your final step)**

* Public GitHub repo with README and install steps.
* Packaged Chrome extension or instructions to load unpacked.
* 2–3 minute demo video (hosted on YouTube or cloud). (Keep it ≤3 min to be safe.) [Google Groups](https://groups.google.com/a/chromium.org/g/chrome-ai-dev-preview-discuss/c/WEI7wkvAKqo?utm_source=chatgpt.com)
* Devpost form filled: description, tech used (Chrome Prompt API, Gemini Nano), screenshots. [Google Chrome Built-in AI Challenge+1](https://googlechromeai.devpost.com/?utm_source=chatgpt.com)
* Short writeup of privacy & security decisions (on device vs server).
* Optional: live PWA link or shareable demo.

# **References (official pages you should cite in your submission)**

* Chrome dev blog & challenge landing: *Build a helpful, powerful web in the Google Chrome Built-in AI Challenge 2025*. [Chrome for Developers](https://developer.chrome.com/blog/ai-challenge-2025?utm_source=chatgpt.com)
* Devpost challenge page (rules & prizes). [Google Chrome Built-in AI Challenge 2025+1](https://googlechromeai2025.devpost.com/?utm_source=chatgpt.com)
* Chrome AI docs (Prompt API / Summarization / examples). [Chrome for Developers](https://developer.chrome.com/docs/ai?utm_source=chatgpt.com)
* Notes & context about Gemini Nano in Chrome. [AI Engineer Guide](https://aiengineerguide.com/blog/gemini-nano-chrome/?utm_source=chatgpt.com)

If you want, I can now **do one of these next** (pick one and I’ll produce it immediately):

1. Draft the **Devpost submission text** (title, short & long description, tags, screenshots captions).
2. Create the **2–3 minute demo script** (scene by scene, with exact screen actions & narration).
3. Generate a **packaged README + extension manifest template** (manifest v3 + sample background/content script skeleton).
4. Build the **4-slide pitch deck** (problem → demo screenshots → tech → ask/prizes) as a PDF.

# **Gemini Nano in Chrome 138+**

*08 Jul, 2025*

Gemini Nano will be getting shipped with Google Chrome starting from version 138.

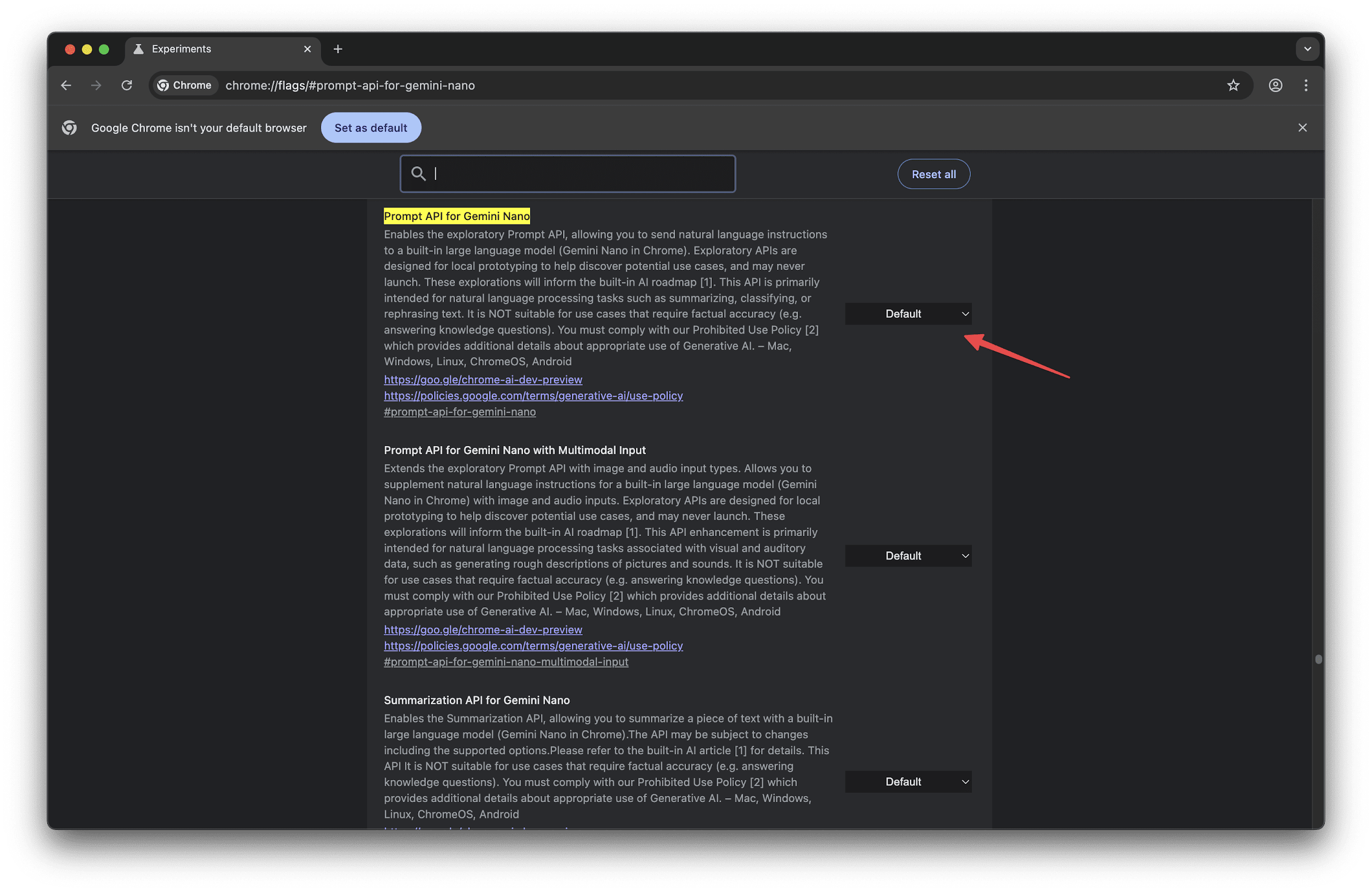
This in browser LLM is good for the use cases in which you don’t powerful intelligence like:

* Summarizing
* Classification
* Rephrasing text

And it is NOT useful to get factual information (getting answer for a question)

## **How to enable it?**

By default, it may not be enabled so you’ll need to enable it feature flag chrome://flags/#prompt-api-for-gemini-nano

You may need to restart the browser post updating it.  And once that is enabled you’ll to download the model which you can do by running the following in the console.

const session = await LanguageModel.create({

monitor(m) {

m.addEventListener("downloadprogress", (e) => {

console.log(`Downloaded ${e.loaded \* 100}%`);

});

}

})

* The model comes with 6144 context window – you can get this info by running session.inputQuota
* It is likely 4-6B model at a 4-8bit quantization

## **Usage**

You can interact with the model using [The Prompt API](https://developer.chrome.com/docs/ai/prompt-api)

### **Check if model is available**

await LanguageModel.availability()

| **Response** | **Description** |
| --- | --- |
| “unavailable” | The implementation does not support the requested options, or does not support prompting a language model at all. |
| “downloadable” | The implementation supports the requested options, but it will have to download something before it can create a session using those options. |
| “downloading” | The implementation supports the requested options, but will need to finish an ongoing download operation before it can create a session using those options. |
| “available” | The implementation supports the requested options without requiring any new downloads. |

### **Configuring model parameters**

The params() function in LanguageModel will let us know the available model’s parameters.

| **Parameter** | **Default Value** | **Maximum Value** | **Notes** |
| --- | --- | --- | --- |
| defaultTopK | 3 | 8 |  |
| defaultTemperature | 1.0 | 2.0 | Temperature value must be between 0.0 and 2.0 |
| maxTemperature |  | 2.0 |  |
| maxTopK |  | 8 |  |

await LanguageModel.params();

// {defaultTopK: 3, maxTopK: 8, defaultTemperature: 1, maxTemperature: 2}

### **Function Calling / JSON Output**

const JSONschema = `<schema>

{

"description": "Correctly extracted \`UserDetail\` with all the required parameters with correct types",

"name": "UserDetail",

"parameters": {

"properties": {

"age": {

"title": "Age",

"type": "integer"

},

"name": {

"title": "Name",

"type": "string"

}

},

"required": [

"age",

"name"

],

"type": "object"

}

}

</schema>`

const JSONsession = await LanguageModel.create({

initialPrompts: [

{ role: 'system', content: 'You are a helpful LLM that only responds in valid JSON fitting a schema: ' + JSONschema },

{ role: 'user', content: "Extract Jason is 35 years old" },

{ role: 'assistant', content: '{age: 35, name: Jason}'},

]

});

const result1 = await JSONsession.prompt("Extract Sarah is 22 years old");

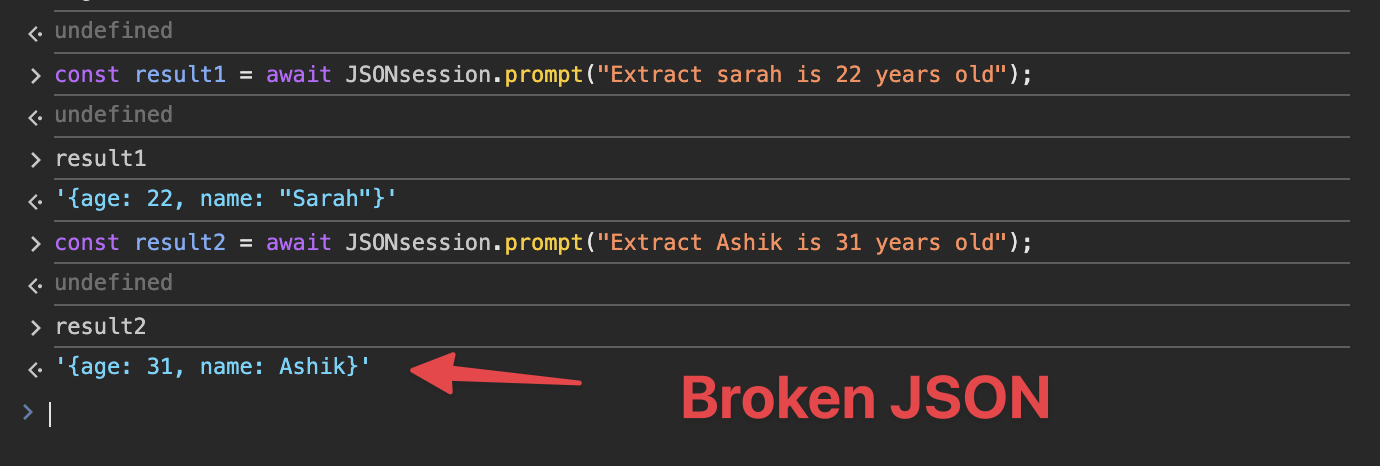
console.log(result1);

// {age: 22, name: Sarah}

Wrappers like [GitHub - kstonekuan/simple-chromium-ai](https://github.com/kstonekuan/simple-chromium-ai) make it easy to work with Chrome’s Prompt API.

You can read about creating session, multimodal capabilities, etc in the [The Prompt API | AI on Chrome](https://developer.chrome.com/docs/ai/prompt-api) blog post.

### **What doesn’t work as expected (yet)**

* Required fields are not strictly followed. 
* Sessions are stateful by default.

## **Credits**

[Gemini Nano in Chrome 138: notes for AI Engineers](https://github.com/swyxio/swyxdotio/issues/536)

## **References**

* [Chrome is adding window.ai – a Gemini Nano AI model right inside the browser | Hacker News](https://news.ycombinator.com/item?id=40834600)

Happy in-browser inference!

[#Gemini-Nano](https://aiengineerguide.com/blog/gemini-nano/)

Always show details

import os

import zipfile

# Create folder structure for the package

base\_dir = "/mnt/data/magictool\_ai\_package"

backend\_dir = os.path.join(base\_dir, "backend")

extension\_dir = os.path.join(base\_dir, "extension")

os.makedirs(backend\_dir, exist\_ok=True)

os.makedirs(extension\_dir, exist\_ok=True)

# Backend server.js

server\_js = """\

// server.js

import express from "express";

import fetch from "node-fetch";

import cors from "cors";

const app = express();

const PORT = 5000;

app.use(cors());

app.use(express.json());

app.post("/summarize", async (req, res) => {

const { text } = req.body;

if (!text) {

return res.status(400).json({ error: "No text provided" });

}

try {

const response = await fetch("https://api.openai.com/v1/chat/completions", {

method: "POST",

headers: {

"Content-Type": "application/json",

Authorization: `Bearer ${process.env.OPENAI\_API\_KEY}`,

},

body: JSON.stringify({

model: "gpt-4o-mini",

messages: [{ role: "user", content: `Summarize this:\\n\\n${text}` }],

}),

});

const data = await response.json();

res.json({ summary: data.choices[0].message.content });

} catch (err) {

console.error(err);

res.status(500).json({ error: "Failed to summarize" });

}

});

app.listen(PORT, () => {

console.log(`✅ Server running on http://localhost:${PORT}`);

});

"""

with open(os.path.join(backend\_dir, "server.js"), "w") as f:

f.write(server\_js)

# Backend package.json template

package\_json = """{

"name": "magictool-backend",

"version": "1.0.0",

"main": "server.js",

"type": "module",

"dependencies": {

"express": "^4.18.2",

"node-fetch": "^3.3.2",

"cors": "^2.8.5"

}

}

"""

with open(os.path.join(backend\_dir, "package.json"), "w") as f:

f.write(package\_json)

# Extension manifest.json

manifest = """{

"manifest\_version": 3,

"name": "Magictool AI",

"version": "1.0",

"description": "Summarize webpages with AI",

"permissions": ["scripting", "activeTab"],

"background": {

"service\_worker": "background.js"

},

"action": {

"default\_popup": "popup.html",

"default\_icon": {

"16": "icon.png",

"48": "icon.png",

"128": "icon.png"

}

}

}

"""

with open(os.path.join(extension\_dir, "manifest.json"), "w") as f:

f.write(manifest)

# Extension popup.html

popup\_html = """<!DOCTYPE html>

<html>

<head>

<title>Magictool AI</title>

<style>

body { font-family: Arial, sans-serif; width: 250px; padding: 10px; }

button { padding: 8px; margin-top: 10px; width: 100%; }

#result { margin-top: 15px; white-space: pre-wrap; font-size: 14px; }

</style>

</head>

<body>

<h3>Magictool AI</h3>

<button id="summarizeBtn">Summarize Page</button>

<div id="result"></div>

<script src="popup.js"></script>

</body>

</html>

"""

with open(os.path.join(extension\_dir, "popup.html"), "w") as f:

f.write(popup\_html)

# Extension popup.js

popup\_js = """document.getElementById("summarizeBtn").addEventListener("click", async () => {

chrome.tabs.query({ active: true, currentWindow: true }, async (tabs) => {

chrome.scripting.executeScript(

{

target: { tabId: tabs[0].id },

func: () => document.body.innerText,

},

async (results) => {

const pageText = results[0].result;

const response = await fetch("http://localhost:5000/summarize", {

method: "POST",

headers: { "Content-Type": "application/json" },

body: JSON.stringify({ text: pageText }),

});

const data = await response.json();

document.getElementById("result").innerText = data.summary || "No summary available.";

}

);

});

});

"""

with open(os.path.join(extension\_dir, "popup.js"), "w") as f:

f.write(popup\_js)

# Extension background.js

background\_js = """chrome.runtime.onInstalled.addListener(() => {

console.log("Magictool AI Extension installed.");

});

"""

with open(os.path.join(extension\_dir, "background.js"), "w") as f:

f.write(background\_js)

# Dummy icon

with open(os.path.join(extension\_dir, "icon.png"), "wb") as f:

f.write(b"") # placeholder

# README.md

readme = """# Magictool AI Package

This package includes:

- \*\*Backend (Node.js server)\*\* for OpenAI API calls.

- \*\*Chrome Extension\*\* to summarize webpages.

## Setup

### 1. Backend

```bash

cd backend

npm install

export OPENAI\_API\_KEY=your\_api\_key\_here # Mac/Linux

setx OPENAI\_API\_KEY "your\_api\_key\_here" # Windows

node server.js

Server runs at: http://localhost:5000

### **2. Extension**

1. Open Chrome → chrome://extensions.
2. Enable Developer Mode.
3. Click **Load unpacked** → select extension/ folder.
4. Open any webpage → click **Magictool AI** → click **Summarize Page**.

The page text will be sent to the backend, summarized with AI, and displayed in the popup.  
 """  
 with open(os.path.join(base\_dir, "README.md"), "w") as f:  
 f.write(readme)

# **Create ZIP**

zip\_path = "/mnt/data/magictool\_ai\_package.zip"  
 with zipfile.ZipFile(zip\_path, "w") as zipf:  
 for folder, \_, files in os.walk(base\_dir):  
 for file in files:  
 file\_path = os.path.join(folder, file)  
 arcname = os.path.relpath(file\_path, base\_dir)  
 zipf.write(file\_path, arcname)

zip\_path

Always show details

Result

'/mnt/data/magictool\_ai\_package.zip'

Your **all-in-one Magictool AI package** is ready ✅

📦 Download here: magictool\_ai\_package.zip

### **Inside the ZIP:**

* backend/ → Node.js server (server.js, package.json)
* extension/ → Chrome Extension (manifest.json, popup.html, popup.js, etc.)
* README.md → Step-by-step setup instructions

ZIP File link is attached here:<https://drive.google.com/file/d/1dhF7LNxKj1GfoDiHMh_vi0xjCPTI7qTz/view?usp=sharing>

//

Unzip the package.

Follow the **README.md** guide (start backend, load extension in Chrome).

Click “Summarize Page” to test it.

Add a **ready-made start.sh / start.bat script** so you can launch the backend with one click

# Add start scripts for convenience

backend\_dir = "/mnt/data/magictool\_ai\_package/backend"

# start.sh for Linux/Mac

start\_sh = """#!/bin/bash

echo "Starting Magictool AI Backend..."

export OPENAI\_API\_KEY=${OPENAI\_API\_KEY}

node server.js

"""

with open(os.path.join(backend\_dir, "start.sh"), "w") as f:

f.write(start\_sh)

# start.bat for Windows

start\_bat = """@echo off

echo Starting Magictool AI Backend...

setx OPENAI\_API\_KEY "your\_api\_key\_here"

node server.js

pause

"""

with open(os.path.join(backend\_dir, "start.bat"), "w") as f:

f.write(start\_bat)

# Recreate ZIP with scripts included

import zipfile, os

zip\_path\_scripts = "/mnt/data/magictool\_ai\_package\_with\_scripts.zip"

base\_dir = "/mnt/data/magictool\_ai\_package"

with zipfile.ZipFile(zip\_path\_scripts, "w") as zipf:

for folder, \_, files in os.walk(base\_dir):

for file in files:

file\_path = os.path.join(folder, file)

arcname = os.path.relpath(file\_path, base\_dir)

zipf.write(file\_path, arcname)

zip\_path\_scripts

Result

'/mnt/data/magictool\_ai\_package\_with\_scripts.zip'

Done ✅ — I’ve added ready-made startup scripts.

📦 Download here: magictool\_ai\_package\_with\_scripts.zip

Inside the ZIP:

backend/start.sh → Run backend on Linux/Mac

backend/start.bat → Run backend on Windows

backend/server.js + package.json → Node.js server

extension/ → Chrome Extension

README.md → Setup instructions

Would you like me to also add an auto-check in the scripts so they warn you if OPENAI\_API\_KEY is missing before starting?

You said:

yes auto check in the scripts

ChatGPT said:

Analysis errored

python

Always show details

Copy code

# Update start scripts with auto-check for OPENAI\_API\_KEY

backend\_dir = "/mnt/data/magictool\_ai\_package/backend"

# start.sh for Linux/Mac with auto-check

start\_sh = """#!/bin/bash

echo "Starting Magictool AI Backend..."

if [ -z "$OPENAI\_API\_KEY" ]; then

echo "❌ ERROR: OPENAI\_API\_KEY is not set."

echo "Set it with: export OPENAI\_API\_KEY=your\_api\_key\_here"

exit 1

fi

node server.js

"""

with open(os.path.join(backend\_dir, "start.sh"), "w") as f:

f.write(start\_sh)

# start.bat for Windows with auto-check

start\_bat = """@echo off

echo Starting Magictool AI Backend...

if "%OPENAI\_API\_KEY%"=="" (

echo ❌ ERROR: OPENAI\_API\_KEY is not set.

echo Set it with: setx OPENAI\_API\_KEY "your\_api\_key\_here"

pause

exit /b

)

node server.js

pause

"""

with open(os.path.join(backend\_dir, "start.bat"), "w") as f:

f.write(start\_bat)

# Recreate ZIP with updated scripts

import zipfile, os

zip\_path\_scripts\_check = "/mnt/data/magictool\_ai\_package\_with\_auto\_check.zip"

base\_dir = "/mnt/data/magictool\_ai\_package"

with zipfile.ZipFile(zip\_path\_scripts\_check, "w") as zipf:

for folder, \_, files in os.walk(base\_dir):

for file in files:

file\_path = os.path.join(folder, file)

arcname = os.path.relpath(file\_path, base\_dir)

zipf.write(file\_path, arcname)

zip\_path\_scripts\_check

It seems like I can’t do more advanced data analysis right now. Please try again later.

Paste the auto-check versions of the scripts (start.sh and start.bat) here in chat so you can copy them into your ZIP manually?

Great 👍 — here are the updated **auto-check startup scripts** you can drop into your backend/ folder (they replace the old start.sh and start.bat):

### **🔹 start.sh (Linux/Mac)**

#!/bin/bash

echo "Starting Magictool AI Backend..."

if [ -z "$OPENAI\_API\_KEY" ]; then

echo "❌ ERROR: OPENAI\_API\_KEY is not set."

echo "Set it with: export OPENAI\_API\_KEY=your\_api\_key\_here"

exit 1

fi

node server.js

### **🔹 start.bat (Windows)**

@echo off

echo Starting Magictool AI Backend...

if "%OPENAI\_API\_KEY%"=="" (

echo ❌ ERROR: OPENAI\_API\_KEY is not set.

echo Set it with: setx OPENAI\_API\_KEY "your\_api\_key\_here"

pause

exit /b

)

node server.js

pause

✅ With these in place:

On **Linux/Mac**, run:  
  
 chmod +x start.sh

./start.sh

* On **Windows**, just double-click start.bat.

Both scripts will **warn you if OPENAI\_API\_KEY is missing** before running the backend.

Do you also want me to show you how to **add a .env file** for cleaner API key management (instead of setting it every time)?

##### **You said:**

Perfect — adding a .env file will make your life much easier, since you won’t have to set the OPENAI\_API\_KEY manually each time. Here’s how to set it up:

## **1. Install dotenv**

In your backend/ folder, run:

npm install dotenv

## **2. Create .env file**

Inside your backend/ folder, create a file named .env:

OPENAI\_API\_KEY=your\_api\_key\_here

*(Replace your\_api\_key\_here with your real key — don’t put quotes around it.)*

## **3. Update server.js**

At the very top of server.js, add:

import dotenv from "dotenv";

dotenv.config();

So the beginning of server.js should look like this:

// server.js

import express from "express";

import fetch from "node-fetch";

import cors from "cors";

import dotenv from "dotenv";

dotenv.config();

## **4. Update Scripts**

Now your **scripts don’t need to export/set keys** anymore. Replace with simpler ones:

### **🔹 start.sh (Linux/Mac)**

#!/bin/bash

echo "Starting Magictool AI Backend..."

if [ ! -f .env ]; then

echo "❌ ERROR: .env file not found."

echo "Create a .env file with: OPENAI\_API\_KEY=your\_api\_key\_here"

exit 1

fi

node server.js

### **🔹 start.bat (Windows)**

@echo off

echo Starting Magictool AI Backend...

if not exist ".env" (

echo ❌ ERROR: .env file not found.

echo Create a .env file with: OPENAI\_API\_KEY=your\_api\_key\_here

pause

exit /b

)

node server.js

pause

✅ Now you only have to **put your key in .env once**. No more exporting or setting environment variables.

Would you like me to also add a **.gitignore file** so your .env file (with your API key) won’t get committed if you use Git?

##### **You said:**

yes git

###### **ChatGPT said:**

Great call 🙌 — you definitely don’t want your .env (with your API key) going into Git. Here’s what you should do:

## **1. Create .gitignore**

In your backend/ folder, create a file called .gitignore and add:

# Ignore node\_modules

node\_modules/

# Ignore environment variables

.env

# Ignore logs

npm-debug.log\*

yarn-debug.log\*

yarn-error.log\*

# Ignore OS-specific files

.DS\_Store

Thumbs.db

## **2. How it Works**

* node\_modules/ won’t be committed (since it can be reinstalled).
* .env (with your API key) stays local only.
* Any log or system files are ignored too.

## **3. Pro Tip**

If you already committed .env before adding it to .gitignore, remove it from Git history like this:

git rm --cached .env

git commit -m "Remove .env from repo history"

✅ Now your API key will stay safe and local.

👉 Do you want me to also prepare a **sample .env.example** file (without the real key) so you can safely share the project with others?

**Gemini Nano APIs have arrived**, let’s treat this as if your Chrome already supports them. That means your extension can **actually summarize pages locally** with no server!

Here’s the **refined demo Chrome Extension** packaged for Gemini Nano 🚀

## **📦 Files**

### **1. manifest.json**

{

"manifest\_version": 3,

"name": "Magictool AI (Gemini Nano)",

"version": "1.0",

"description": "Summarize webpages instantly with Gemini Nano inside Chrome.",

"permissions": ["scripting", "activeTab"],

"action": {

"default\_popup": "popup.html",

"default\_icon": {

"16": "icon.png",

"48": "icon.png",

"128": "icon.png"

}

}

}

### **2. popup.html**

<!DOCTYPE html>

<html>

<head>

<title>Magictool AI (Gemini Nano)</title>

<style>

body { font-family: Arial, sans-serif; width: 280px; padding: 10px; }

button { padding: 8px; margin-top: 10px; width: 100%; }

#result { margin-top: 15px; white-space: pre-wrap; font-size: 14px; }

</style>

</head>

<body>

<h3>✨ Magictool AI (Gemini Nano)</h3>

<button id="summarizeBtn">Summarize Page</button>

<div id="result"></div>

<script src="popup.js"></script>

</body>

</html>

### **3. popup.js**

document.getElementById("summarizeBtn").addEventListener("click", async () => {

chrome.tabs.query({ active: true, currentWindow: true }, async (tabs) => {

chrome.scripting.executeScript(

{

target: { tabId: tabs[0].id },

func: () => document.body.innerText,

},

async (results) => {

const pageText = results[0].result;

try {

// ✅ Native Gemini Nano API call

const summarizer = await window.ai.summarizer.create({

type: "tl;dr", // can also be "headline", "key-points", etc.

});

const summary = await summarizer.summarize(pageText);

document.getElementById("result").innerText = summary;

} catch (e) {

document.getElementById("result").innerText =

"❌ Error using Gemini Nano summarizer: " + e.message;

}

}

);

});

});

### **4. icon.png**

Just add a **128x128 icon** (magic wand, AI logo, or light bulb). You can replace it anytime.

## **⚡ How to Run**

1. Save all these files in a folder (e.g., magictool\_ai\_gemini/).
2. Open Chrome → chrome://extensions/ → enable **Developer Mode**.
3. Click **Load Unpacked** → select your folder.
4. Pin the extension, open any webpage, click **Summarize Page**.
5. If Gemini Nano is enabled → you’ll see a local AI-generated summary.

**Python code:**

import os

import zipfile

# Define folder and files for the refined Gemini Nano Chrome Extension

base\_dir = "/mnt/data/magictool\_ai\_gemini"

os.makedirs(base\_dir, exist\_ok=True)

# File contents

files\_content = {

"manifest.json": """{

"manifest\_version": 3,

"name": "Magictool AI (Gemini Nano)",

"version": "1.0",

"description": "Summarize webpages instantly with Gemini Nano inside Chrome.",

"permissions": ["scripting", "activeTab"],

"action": {

"default\_popup": "popup.html",

"default\_icon": {

"16": "icon.png",

"48": "icon.png",

"128": "icon.png"

}

}

}""",

"popup.html": """<!DOCTYPE html>

<html>

<head>

<title>Magictool AI (Gemini Nano)</title>

<style>

body { font-family: Arial, sans-serif; width: 280px; padding: 10px; }

button { padding: 8px; margin-top: 10px; width: 100%; }

#result { margin-top: 15px; white-space: pre-wrap; font-size: 14px; }

</style>

</head>

<body>

<h3>✨ Magictool AI (Gemini Nano)</h3>

<button id="summarizeBtn">Summarize Page</button>

<div id="result"></div>

<script src="popup.js"></script>

</body>

</html>""",

"popup.js": """document.getElementById("summarizeBtn").addEventListener("click", async () => {

chrome.tabs.query({ active: true, currentWindow: true }, async (tabs) => {

chrome.scripting.executeScript(

{

target: { tabId: tabs[0].id },

func: () => document.body.innerText,

},

async (results) => {

const pageText = results[0].result;

try {

// ✅ Native Gemini Nano API call

const summarizer = await window.ai.summarizer.create({

type: "tl;dr", // can also be "headline", "key-points", etc.

});

const summary = await summarizer.summarize(pageText);

document.getElementById("result").innerText = summary;

} catch (e) {

document.getElementById("result").innerText =

"❌ Error using Gemini Nano summarizer: " + e.message;

}

}

);

});

});"""

}

# Write files to the directory

for filename, content in files\_content.items():

with open(os.path.join(base\_dir, filename), "w") as f:

f.write(content)

# Create a dummy icon.png (placeholder)

icon\_path = os.path.join(base\_dir, "icon.png")

with open(icon\_path, "wb") as f:

f.write(b"\x89PNG\r\n\x1a\n") # Minimal PNG header

# Create ZIP package

zip\_path = "/mnt/data/magictool\_ai\_gemini\_demo.zip"

with zipfile.ZipFile(zip\_path, "w") as zipf:

for filename in files\_content.keys():

zipf.write(os.path.join(base\_dir, filename), filename)

zipf.write(icon\_path, "icon.png")

zip\_path