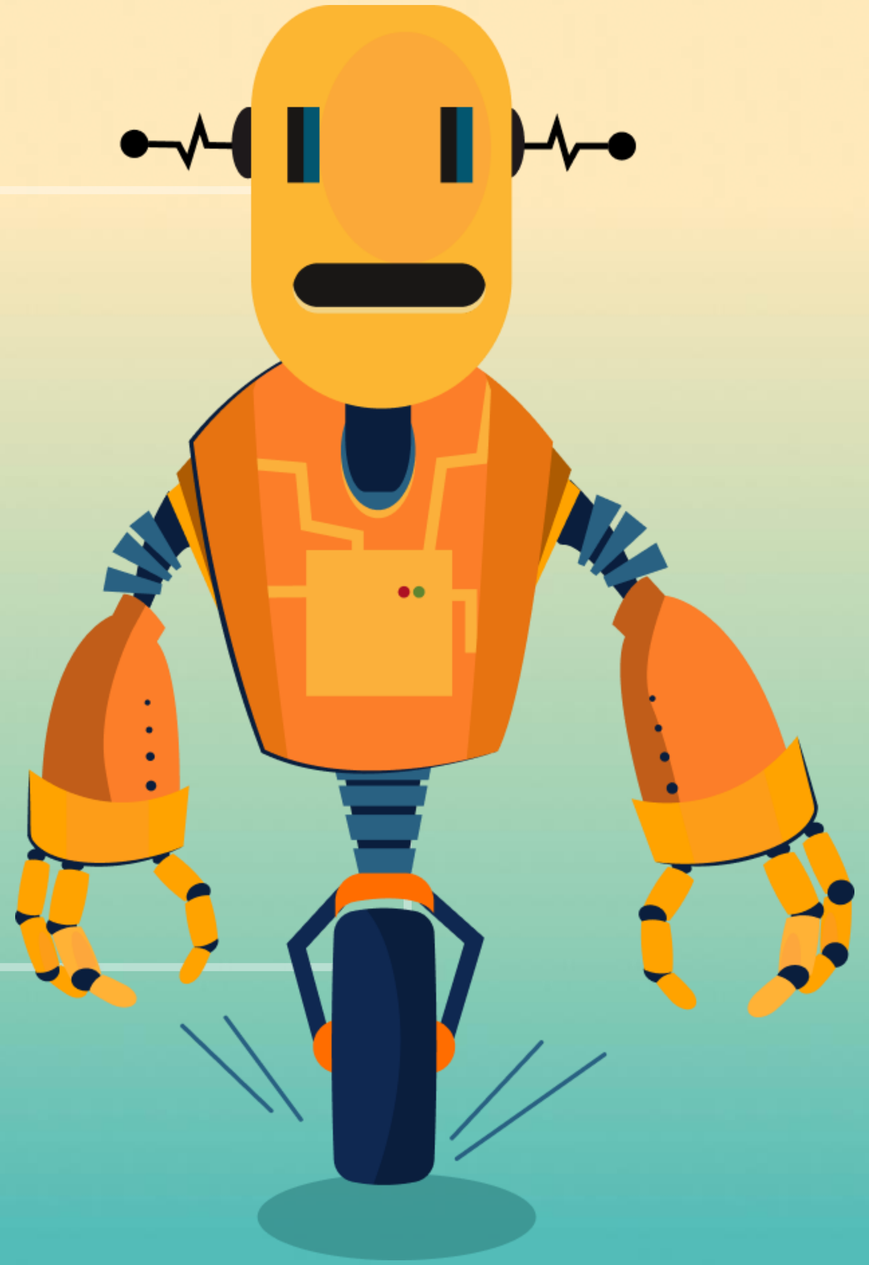
Web AI

Browser-Powered Intelligence



EaTemp

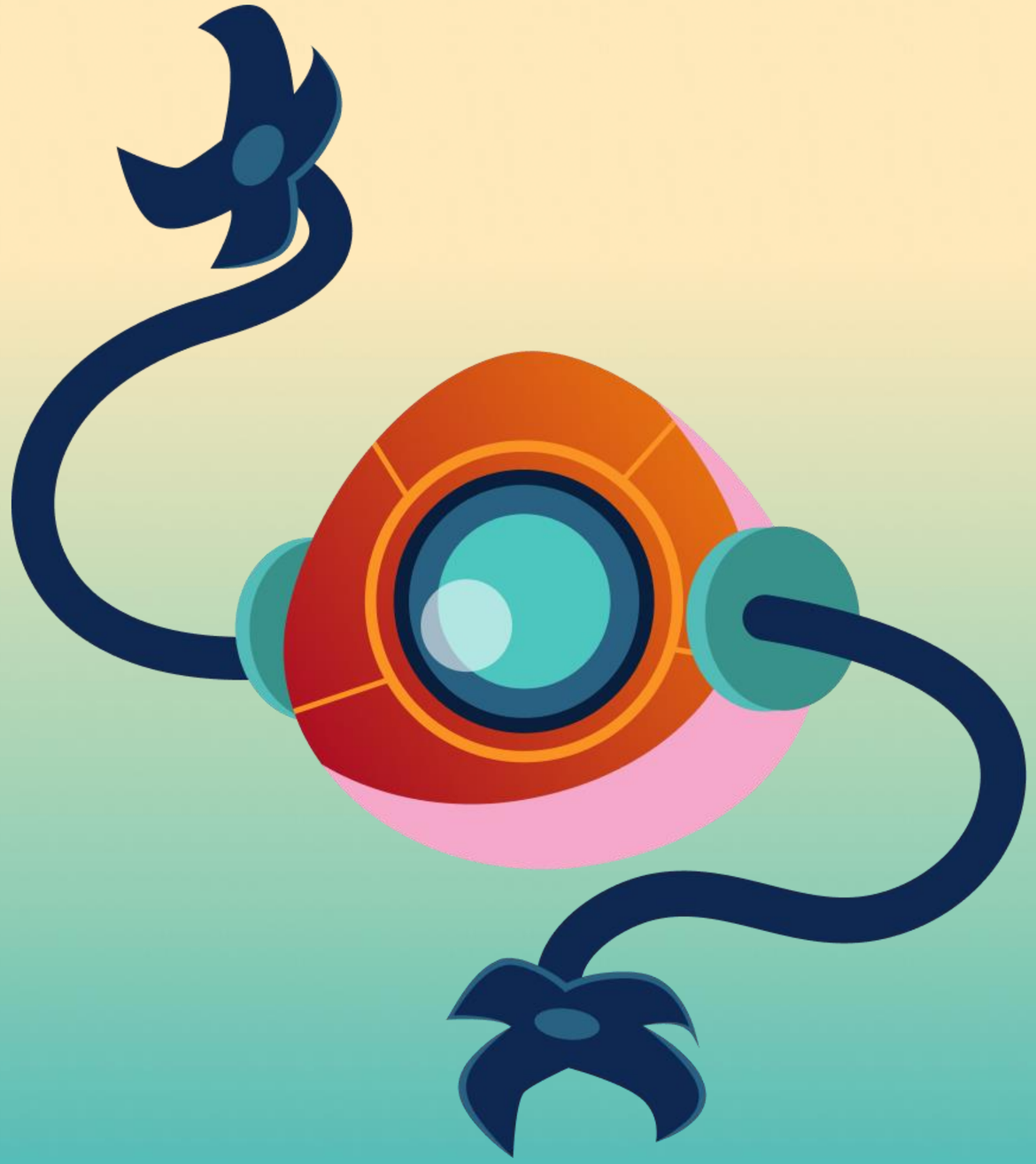
Web AI

Browser-Powered Intelligence

Ron Dagdag

R&D Engineering Manager at 7-ELEVEN

@rondagdag



WELCOME

EaTemp

Agenda

- Server-Side vs Client-Side AI
- Running AI Models in the browser
- Small Language Models
- Built-in AI in the browser
- Demo

EaTemp

Generational Marketing

Generational Marketing

Marketing

Silent Generation	1928–1945
Baby Boomers	1946–1964
Generation X	1965–1980
Millennials	1981–1996
Generation Z	1997–2012

Generational Marketing

The Greatest Generation

Built resilience through the Great Depression and defined heroism in WWII.

1901–1924

The Silent Generation

Hardworking, disciplined, and shaped by post-war stability.

1925–1945

The Baby Boomer Generation

Grew up in prosperity, fueled cultural revolutions, and reshaped work and retirement.

1946–1964

Generation X

The independent, skeptical, latchkey kids who bridged analog and digital worlds.

1965–1979

Millennials

Tech-savvy, purpose-driven, and shaped by 9/11, social media, and the gig economy.

1980–1994

Generation Z

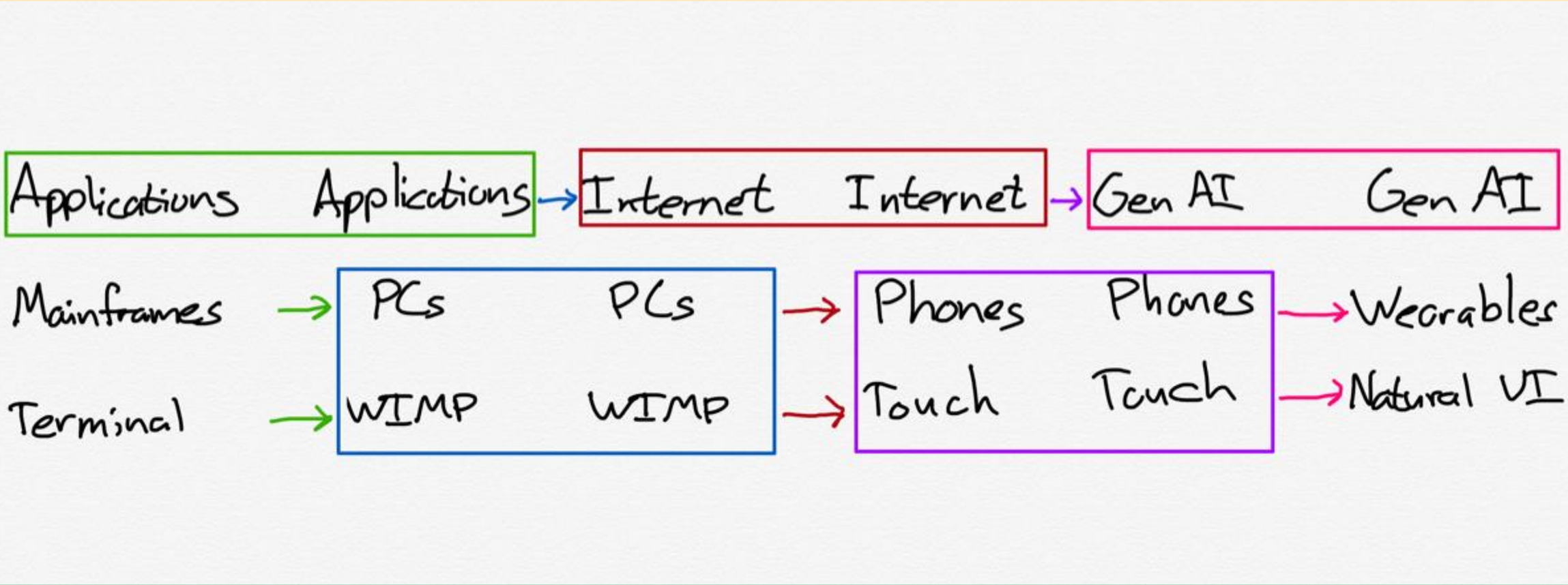
Digital natives, socially conscious, and redefining norms in work, activism, and identity.

1995–2012

Gen Alpha

touchscreen-first generation growing up in an always-connected world.

2013–2025



https://stratechery.com/2024/the-gen-ai-bridge-to-the-future/

Developer Generations

Mainframe Generation (1950s-1970s)	Client-Server Generation (1980s-1990s)	Web Generation (1995-2005)	Mobile Generation (2006-2015)	Cloud Generation (2010-2020)	AI Generation (2020-Present)	Agentic Generation (2025-Future)
Developers who worked with punch cards, COBOL, and Fortran on massive centralized systems.	Developers building networked applications, databases, and enterprise systems using languages like C, C++, and early Java.	Developers who embraced HTML, CSS, JavaScript, and PHP, shaping the first wave of dynamic websites.	Developers who pioneered smartphone apps with iOS, Android, and cross-platform tools like React Native.	Developers focused on scalable, serverless, and distributed computing with AWS, Azure, and DevOps culture.	Developers working with machine learning, LLMs, and AI-powered applications, integrating intelligence into everyday software.	Developers focusing on autonomous AI agents, multi-agent systems, and self-improving software architectures.

Choose Sides

Server-Side

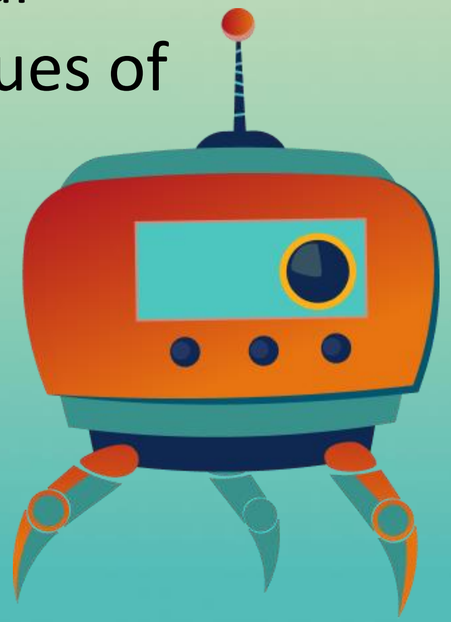
- Supports complex models
- Broader device compatibility
- High performance, scalable, continuous updates
- Leverages the Cloud AI ecosystem like Azure AI Foundry

Client-Side

- Protects data, improves latency, offline support
- Purpose-built models for quick responses
- Model download or built-in browser AI

SMALL LANGUAGE MODELS (SLM)

- subset of language models
- scaled-down variant of a large language model (LLM)
- leveraging architectural principles and techniques of LLMs



- reduction in model size decreases complexity
- compact and efficient
- significantly reduced computational footprint.

efficient in memory usage, computational requirements

EaTemp

Small Language Models (SLMs)

Cost Effectiveness

Deployment Flexibility

Ultra-low Latency

Easier to Customize

SMALL LANGUAGE MODELS

Text Generation

- Creating coherent and contextually relevant sentences or paragraphs

Text Completion

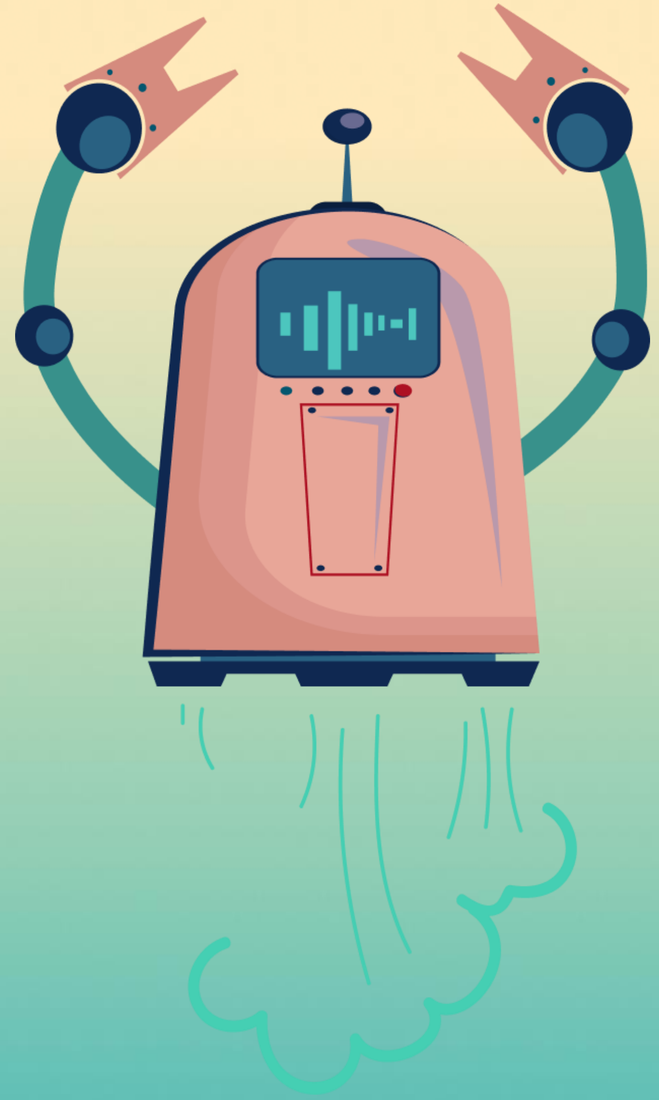
- Predicting and completing sentences based on prompt.

Translation

- Converting text from one language to another.

Summarization

- Condensing long pieces of text into shorter, digestible summaries.



APPLICATIONS

Chatbots	• Providing customer support and engaging with users in a conversational manner.
Content Creation	• Assisting writers by generating ideas or even drafting entire articles.
Education	• Helping students with writing assignments or learning new languages.
Accessibility	• Creating tools for individuals with disabilities, such as text-to-speech systems.

CONSIDERATIONS

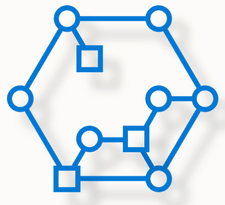
Size	• ChatGPT (GPT-4), = 1.76 trillion parameters • Mistral 7B = 7 billion. • Phi3.5 mini = 3.8 billion / Phi3.5 small = 7 billion
Comprehension	• highly specialized • limited in broad contextual understanding across multiple fields of knowledge
Computing	• LLM training and deployment -> resource-intensive processes • SLM training and deployment -> local machines equipped with good GPU. Takes hours to train
Bias	• Bias is a known issue in LLMs, nature of the training data. • SLMs, trained on domain-specific datasets, are less bias (fine-tuned)
Inference	• inference speed • outputs efficiently on local hardware without extensive parallel processing

Phi

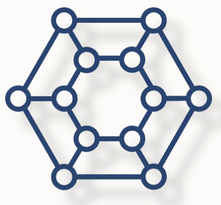
Small Language Models

Groundbreaking performance for size, with frictionless availability

Phi-3.5-mini
(3.8B)



Phi-3.5-vision
(4.2B)



Phi-3.5-MoE
(6.6B active)

Instruction Tuned

RAI Safety Aligned

Available on

Azure AI
Model Catalog

Hugging Face

ONNX Runtime

NVIDIA NIM

Ollama

DEMO

```
/**
 * This class uses the Singleton pattern to enable lazy-loading of the pipeline
 */
class TextGenerationPipeline {
  static model_id = "onnx-community/Phi-3.5-mini-instruct-onnx-web";

  static async getInstance(progress_callback = null) {
    this.tokenizer ??= AutoTokenizer.from_pretrained(this.model_id, {
      progress_callback,
    });

    this.model ??= AutoModelForCausalLM.from_pretrained(this.model_id, {
      // dtype: "q4",
      dtype: "q4f16",
      device: "webgpu",
      use_external_data_format: true,
      progress_callback,
    });

    return Promise.all([this.tokenizer, this.model]);
}
}``
```

```
async function load() {
  self.postMessage({
    status: "loading",
    data: "Loading model...",
  });

  // Load the pipeline and save it for future use.
  const [tokenizer, model] = await TextGenerationPipeline.getInstance((x) => {
    // We also add a progress callback to the pipeline so that we can
    // track model loading.
    self.postMessage(x);
  });

  self.postMessage({
    status: "loading",
    data: "Compiling shaders and warming up model...",
  });

  // Run model with dummy input to compile shaders
  const inputs = tokenizer("a");
  await model.generate({ ...inputs, max_new_tokens: 1 });
  self.postMessage({ status: "ready" });
}
```

Gemini Nano

two small models:

Nano-1

and the slightly more capable

Nano-2

which is meant to run offline.

unlocks new possibilities for AI agents - intelligent systems that can use memory, reasoning, and planning to complete tasks for you.

Built-In AI

- AI models built directly into web browsers
- Runs locally on devices, reducing dependency on servers
- Enhances privacy and performance

Built-In AI

Ease of Deployment	• Browser distributes & updates models • No need to manage large downloads
Hardware Acceleration	• Optimized for GPUs, NPUs, CPUs
Local Processing	• Improves privacy & speeds up responses
Offline Usage	• AI features available without internet

Built-In AI

Privacy & Security	•Sensitive data stays on device
Snappy User Experience	•Near-instant results without server round trips
Cost & Scalability	•Offloads processing to users' devices
Enhanced Feature Access	•Preview premium features with minimal cost

Built-In AI

- **Available APIs:**
  - **Prompt API**
  - **Summarizer API**
  - **Writer/Rewriter API**
  - **Translator API**
  - **Language Detector API**

- **Note:**
  - **All download Gemini Nano (text-to-text only)**
  - **Not available on mobile devices**

Built-In AI

API	Explainer	Web	Extensions	Chrome Status	Intent
Translator API	GitHub	Origin trial	Origin trial	View	Intent to Experiment
Language Detector API	GitHub	Origin trial	Origin trial	View	Intent to Experiment
Summarizer API	GitHub	Origin trial	Origin trial	View	Intent to Experiment
Writer API	GitHub	In EPP	In EPP	View	Intent to Prototype
Rewriter API	GitHub	In EPP	In EPP	View	Intent to Prototype
Prompt API	GitHub	In EPP	Origin trial	Not applicable	Not applicable

Built-In AI

Hardware & OS Requirements

Supported OS:

- Windows 10/11, macOS 13+ (Ventura+), Linux

Storage:

- Minimum 22 GB on Chrome profile volume
- Model requires only a couple of GB, but ample space is needed

GPU:

- More than 4 GB of VRAM required

Network:

- Unlimited/unmetered connection (avoid cellular metered connections)

Built-In AI

Installation Steps

- First of all, you'll need to download Chrome Canary

- In chrome://flags, you must **enable** two experiments:

  - Prompt API for Gemini Nano
  - Enables optimization guide on device.

- Restart the browser, after having enabled those two flags.

Wait, until it downloads Gemini Nano
(1.7GB of space, need about 20GB at installation time) but the API will tell you if the model weights are not fully downloaded yet.

Built-In AI

Verification Steps

1. Open DevTools and send `(await ai.languageModel.capabilities()).available;` in the console.
2. If this returns `"readily"`, then you are all set.

If this fails, continue as follows:

1. 📣🆕 Force Chrome to recognize that you want to use this API. To do so, open DevTools and send `await ai.languageModel.create();` in the console. This will likely fail but it's intended.
2. Relaunch Chrome.
3. Open a new tab in Chrome, go to `chrome://components`
4. Confirm that Gemini Nano is either available or is being downloaded
    - You'll want to see the `Optimization Guide On Device Model` present with a **version greater or equal to `2024.5.21.1031`.**
    - If there is no version listed, click on `Check for update` to force the download.
5. Once the model has downloaded and has reached a version greater than shown above, open DevTools and send `(await ai.languageModel.capabilities()).available;` in the console. If this returns `"readily"`, then you are all set.
    - Otherwise, relaunch, wait for a little while, and try again from step 1.

Built-In AI

Not yet supported:

- Chrome for Android
- Chrome for iOS
- Chrome for ChromeOS

AI in Chrome Extensions

Enhance Browsing	•Control and customize web content
Elevate User Experience	•Smarter bookmarks, new tab pages, history insights
Seamless Integration	•Side panels, action bars, context menus

Hybrid AI Approach

Client-Side	•Handles specific, approachable tasks
Server-Side	•For complex models & broader device compatibility
Graceful Fallback	•Ensures functionality on older or less powerful devices

Explain in Generations

```
// Extension installation listener
chrome.runtime.onInstalled.addListener(() => {
  chrome.contextMenus.create({
    id: "summarize-text",
    title: "Explain in Generations",
    contexts: ["selection"]
  })
})

const getOptions = () => ({
  sharedContext: `${currentLevel.context}. ${currentLevel.description}`,
  type: "tl;dr",
  format: "plain-text",
  length: "medium"
})
```

```typescript
const levels: Level[] = [
  {
    level: 7,
    name: "Generation Alpha",
    context: "Explain like I'm from Generation Alpha (2013-2025). Use emojis 🚀, gamification elements, and frequent emoji reactions. Break into mini-challenges, use AI/tech analogies, and keep it super interactive. Think: TikTok/YouTube-style explanations with quick transitions and visual cues.",
    description: "Ultra-interactive digital native style 🎮, gamified learning chunks 🎯, AI-friendly explanations 🤖, emoji-rich communication 😊"
  },
  {
    level: 6,
    name: "Generation Z",
    context: "Explain like I'm from Generation Z (1995-2012). Use social media inspired formats, internet slang, and cultural references. Keep it real and skip corporate speak. Think: Instagram captions, TikTok scripts, or Twitter threads. Include occasional memes and focus on quick, memorable takeaways.",
    description: "Social media style explanations, internet culture references, authenticity over formality, meme-friendly format"
  },
  {
    level: 5,
    name: "Millennial",
    context: "Explain like I'm a Millennial (1980-1994). Reference 90s/2000s pop culture, use nostalgic comparisons, and blend humor with adulting wisdom. Think: BuzzFeed-style lists, Friends references, Harry Potter analogies. Balance fun with practical life hacks and side-hustle mindset.",
    description: "90s/2000s cultural references, witty yet practical, startup/side-hustle mindset, list-style breakdowns"
  },
  {
    level: 4,
    name: "Generation X",
    context: "Explain like I'm from Generation X (1965-1979). Be direct and slightly cynical, avoid corporate buzzwords. Use references to classic rock, early tech, or 80s culture when relevant. Think: MTV generation meets pragmatic problem-solving. Focus on independence and getting things done efficiently.",
    description: "Straight-to-the-point, DIY approach, healthy skepticism, 80s cultural touchstones, values resourcefulness"
  },
```

```
  {
    level: 3,
    name: "Baby Boomer",
    context: "Explain like I'm a Baby Boomer (1946-1964). Use post-war and economic boom references, emphasize traditional
    values and hard work. Think: newspaper article style, Walter Cronkite-era clarity. Include historical context and draw
    parallels to major events from 50s-70s. Value experience and proven track records.",
    description: "Traditional media style, historical context heavy, values-based reasoning, experience-focused explanations"
  },
  {
    level: 2,
    name: "The Silent Generation",
    context: "Explain like I'm from the Silent Generation (1925-1945). Use formal, respectful language with clear hierarchy.
    Reference post-depression era values, military precision, and methodical approaches. Think: formal letter or professional
    memo style. Emphasize duty, honor, and careful planning.",
    description: "Highly structured formal style, military-precise language, duty-oriented perspective, traditional values
    focus"
  },
  {
    level: 1,
    name: "The Greatest Generation",
    context: "Explain like I'm from the Greatest Generation (1901-1924). Use very formal, authoritative language with proper
    etiquette. Reference early 20th century contexts, classical literature, and time-tested principles. Think: formal academic
    lecture or professional correspondence style. Focus on foundational wisdom and proven methodologies.",
    description: "Classical formal style, scholarly tone, foundational principles, traditional wisdom emphasis"
  }
];
```

```
try {
  // @ts-expect-error new chrome feature
  const available = (await self.ai.summarizer.capabilities()).available
  let summarizer
  if (available === "no") {
    chrome.runtime.sendMessage({
      type: "ERROR",
      error: "The Summarizer API isn't usable"
    })
    return
  }
  if (available === "readily") {
    chrome.runtime.sendMessage({
      chunk: "",
      type: "STREAM_RESPONSE",
      isFirst: true,
      level: currentLevel.level
    })
    // @ts-expect-error new chrome feature
    summarizer = await self.ai.summarizer.create(getOptions())

    await summarizer.ready

    const stream = await summarizer.summarize(info.selectionText, {
      context: `article from ${new URL(tab.url!).origin}`
    })
    for await (const chunk of stream) {
      chrome.runtime.sendMessage({
        chunk,
        type: "STREAM_RESPONSE"
      })
    }
    chrome.runtime.sendMessage({
      type: "STREAM_COMPLETE"
    })
  } else {
```

QUESTIONS

EaTemp

Ron Dagdag

- R&D Engineering Manager at 7-Eleven
- 9th year Microsoft MVP awardee
- www.dagdag.net
- @rondagdag
- Linked In
  www.linkedin.com/in/rondagdag/
- Thanks for geeking out with me about Web AI