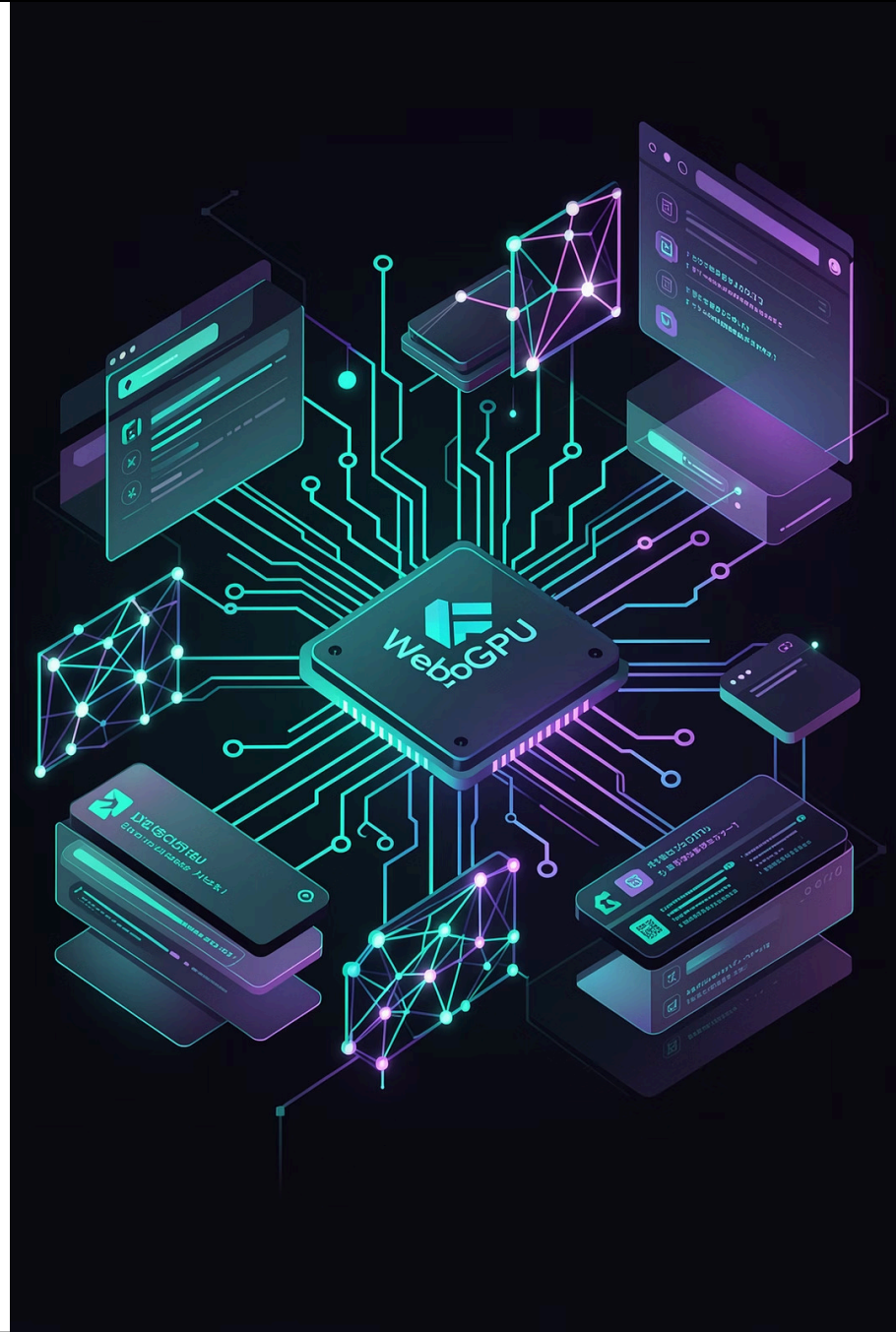
Gemma-gem 기술 상세 분석

브라우저 안에서 동작하는 온디바이스 Gemma 4 에이전트 확장

CHROME EXTENSION | OFFSCREEN RUNTIME | WEBGPU GEMMA 4 | TOOL-USING AGENT

www.deformatic.ai.kr 유민수 개발자



이 발표의 목적

이 자료는 Gemma-gem 저장소를 기술적으로 해부한다. 단순 소개가 아니라 **구현 레벨**에서 파일, 모듈, 메시지, 런타임 역할을 직접 추적한다. 무엇을 해결하려는 프로젝트인지, 어떻게 구성되어 있는지, 왜 이런 구조를 선택했는지 설명한다.

→ **무엇을**

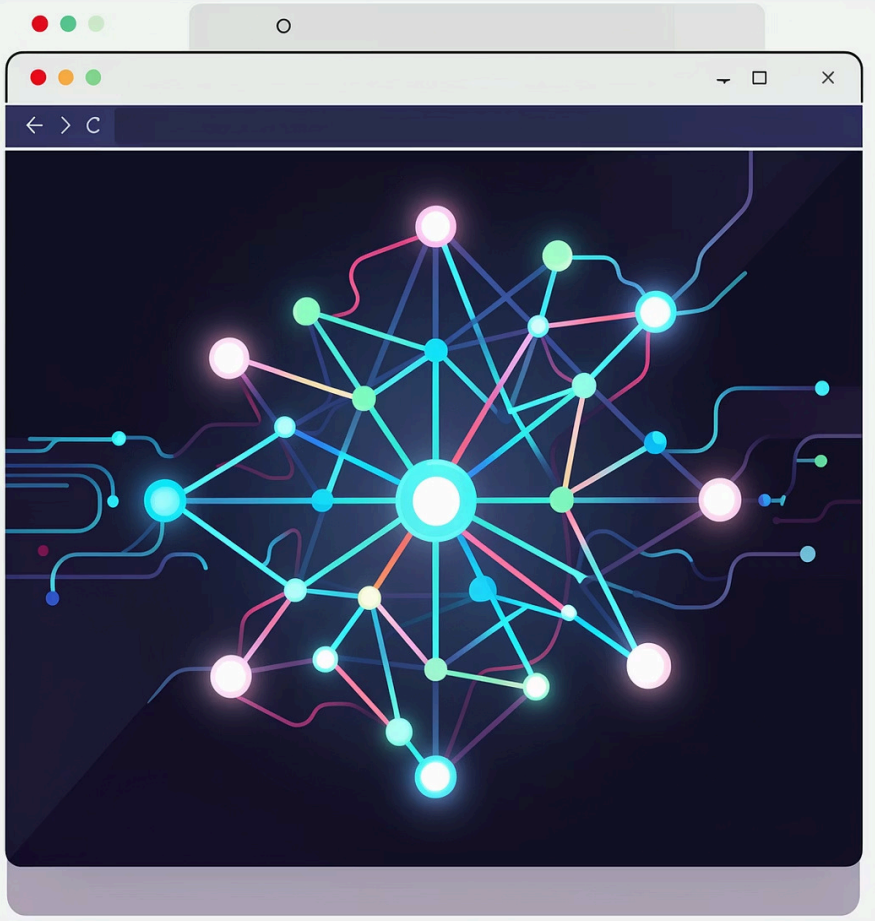
어떤 문제를 해결하는 프로젝트인가

→ **어떻게**

파일, 모듈, 런타임 역할을 추적

→ **왜**

이런 구조를 선택한 설계 근거



Gemma-gem을 한 문장으로 정의하면

> 브라우저 확장 프로그램 안에서 Gemma 4를 온디바이스로 실행하면서, 현재 열려 있는 웹페이지를 읽고 조작할 수 있게 만든 **로컬 AI 에이전트 구현**이다.

browser-native

서버 없이 브라우저 안에서 완결

on-device inference

WebGPU 기반 로컬 추론

agent loop

tool call 기반 반복 실행

DOM tools

페이지 읽기·클릭·입력·스크롤

이 저장소가 흥미로운 이유

브라우저 제약 안에서 LLM + 에이전트 동시 성립

브라우저 확장이라는 극히 제한된 실행 환경에서 LLM 로컬 추론과 툴 호출형 에이전트를 하나의 코드베이스로 구현한다. 이 두 가지를 동시에 성립시키는 것 자체가 설계 도전이다.

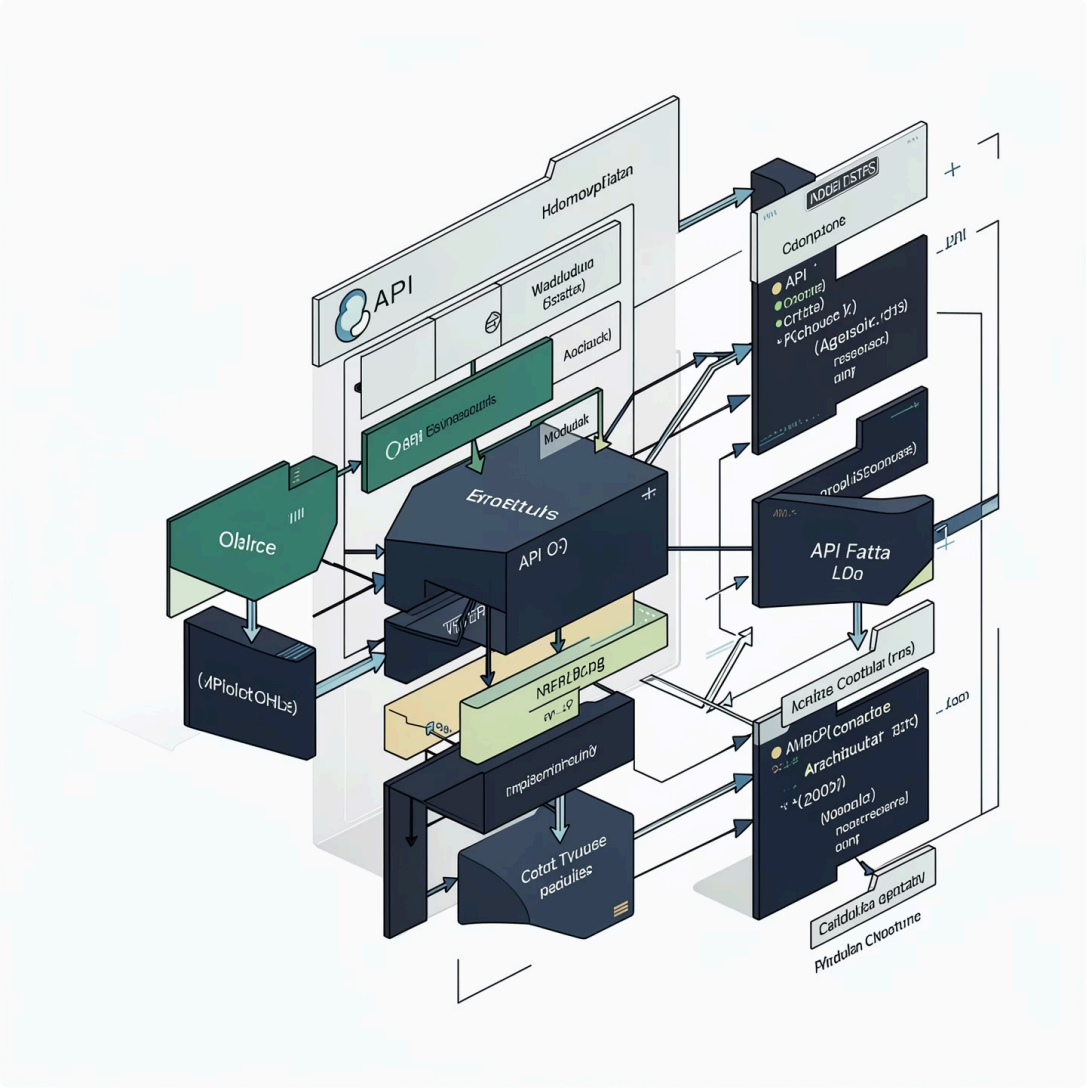
서버 없는 WebGPU 기반 로컬 추론

외부 API나 원격 서버 호출 없이 WebGPU를 통해 브라우저 내부에서 직접 모델을 실행한다. 네트워크 의존성이 없는 진정한 온디바이스 추론 파이프라인이다.

현재 탭에 밀착한 page context 주입

현재 열려 있는 탭을 대상으로 page snapshot을 주입하고, 필요 시 CSS selector 기반 도구를 호출해 페이지를 실시간으로 조작한다. 범용 챗봇이 아닌 페이지 밀착형 에이전트다.

제품이 아니라 기술 레퍼런스로 볼 때의 가치



1 응축된 소형 코드베이스

핵심 개념이 작은 파일 수에 밀집되어 있다. 전체 흐름을 빠르게 파악할 수 있다.

2 관심사 명확한 분리

모델 추론, 브라우저 권한, DOM 조작, UI 스트리밍이 각 파일과 컨텍스트로 분리되어 있다.

3 브라우저 에이전트 설계 학습용

실전 구현이 작동하는 방식을 직접 읽으면서 따라갈 수 있는 최적의 레퍼런스다.

저장소가 직접 내세우는 핵심 개념

Gemma 4, 브라우저 안에서 실행

Gemma 4 모델이 서버 없이 브라우저 내 WebGPU를 통해 로컬로 실행된다. 인터넷 연결 없이도 추론이 가능하다.

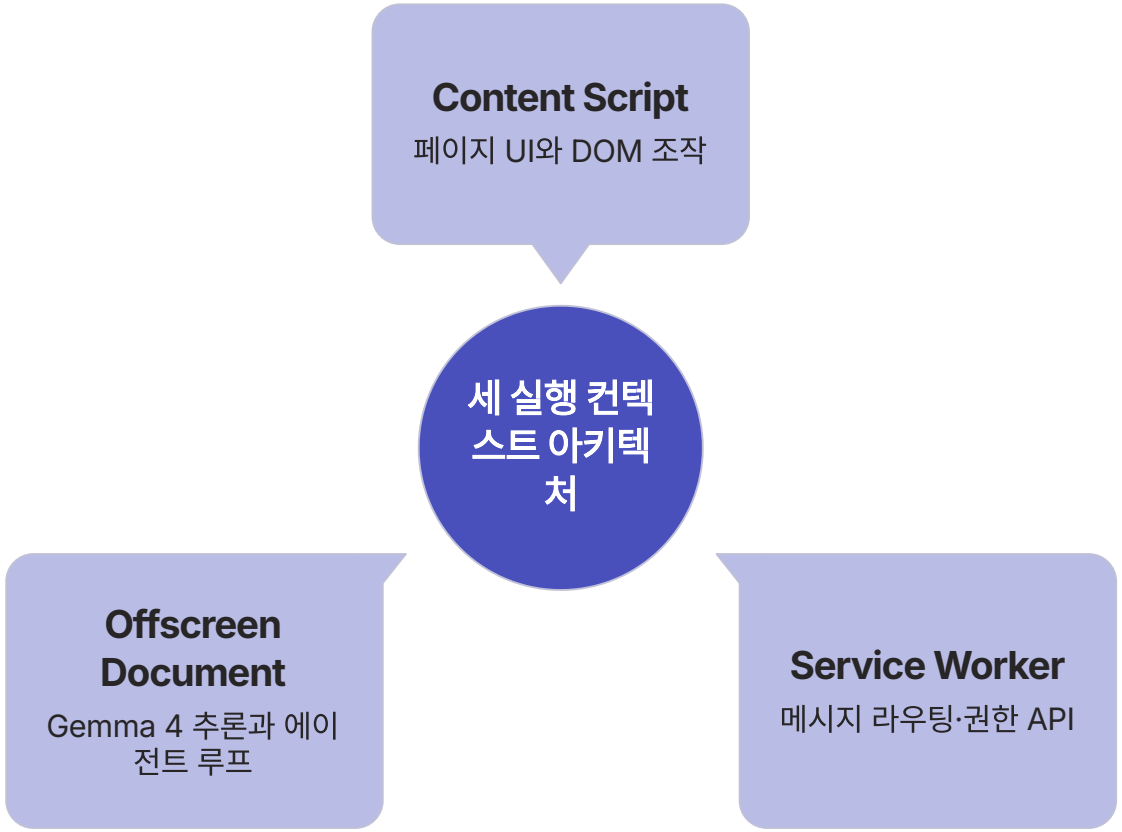
프라이버시 지향, 데이터 로컬 처리

사용자 입력과 페이지 데이터가 외부로 전송되지 않는다. 로컬 추론 구조가 이를 구조적으로 보장한다.

브라우저 에이전트 조작 능력

페이지 읽기, 클릭, 텍스트 입력, 스크롤, 스크린샷, JavaScript 직접 실행까지 광범위한 페이지 조작이 가능하다.

전체 구조 개요: 세 개의 실행 컨텍스트



Chrome 확장의 핵심은 세 실행 컨텍스트의 분리다. **Content Script**는 페이지 위 UI와 DOM 조작, **Service Worker**는 메시지 라우팅과 privileged API 접근, **Offscreen Document**는 Gemma 4 추론과 agent loop를 각각 담당한다. 이 3분할이 전체 설계의 근간이다.

왜 이 3분할 구조가 중요한가

Content Script

DOM 접근이 가장 자연스럽고 직접적

Service Worker

captureVisibleTab, scripting 등 privileged API 전담

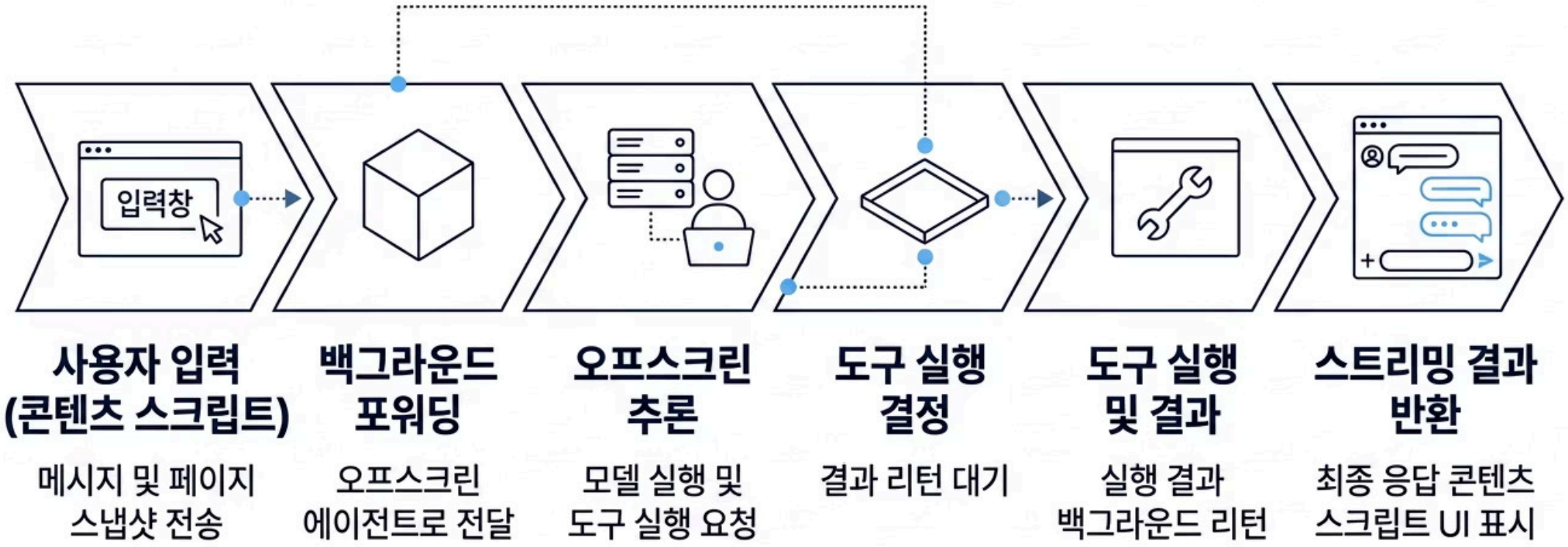
Offscreen Document

무거운 모델 추론 격리, UI·권한 로직 단순화

ⓘ Chrome extension은 모든 권한과 DOM 접근을 하나의 컨텍스트에 몰아둘 수 없다. 각 런타임의 제약을 받아들이고 역할을 분리하는 것이 이 구조의 핵심 설계 원칙이다.

모델 추론을 offscreen으로 격리함으로써 service worker와 content script 각각의 코드가 단순해지고, 권한 위반 위험이 줄어든다.

아키텍처를 데이터 흐름으로 보면



전체 데이터 흐름은 사용자 입력에서 출발해 컨텍스트를 가로질러 돌아온다. **page snapshot**이 함께 전달되어 모든 질문에 페이지 맥락이 자동으로 포함된다. tool call이 발생하면 background 또는 content script가 실행하고 결과를 다시 offscreen으로 반환한다.

저장소 구조 개요

Entrypoints

```
entrypoints/
  background.ts
  content.ts
  offscreen/
    main.ts
```

Background

```
background/
  offscreen-manager.ts
  message-router.ts
```

Offscreen / Content / Shared

```
offscreen/
  model-host.ts
content/
  tool-executors.ts
shared/
  messages.ts
```

Config

```
wxt.config.ts
package.json
```

파일 수는 적지만 각 파일이 명확한 책임을 가진다. 이 구조 자체가 관심사 분리 원칙의 직접적인 반영이다.

기술 스택

WXT

브라우저 확장 개발 프레임워크. manifest, entrypoint, service worker, offscreen 구성 통합 관리.

@huggingface/transformers

브라우저 내 추론 계층. Gemma 4 모델 로딩과 생성 API를 제공한다.

ONNX Runtime Web

실제 모델 연산 백엔드. WebGPU와 wasm 양쪽을 지원한다.

@kessler/gemma-agent

agent orchestration 계층. tool definition 연결과 loop 관리.

marked

채팅 응답 렌더링용 마크다운 처리 라이브러리.

왜 WXT를 썼는가

WXT가 해결하는 문제

브라우저 확장 특유의 번들링·배포 설정 부담을 줄여준다. manifest, content script, service worker, offscreen entry를 구조적으로 일관되게 관리할 수 있어 설정 오류 위험이 낮아진다.

→ 단순한 entrypoint 구성

파일 시스템 기반 entrypoint 자동 인식으로 manifest 직접 편집 최소화

→ 확장 특화 번들링

content script isolation, background 번들 분리를 자동 처리

→ 개발 생산성

HMR, 타입 지원, 확장 리로드 자동화로 개발 사이클 단축

package.json이 보여주는 설계 방향

> **"Browser AI agent powered by Gemma 4 via WebGPU"**

패키지 설명 한 줄이 프로젝트 의도를 명확히 말한다. 단순 챗봇 확장이 아니라 **브라우저 조작 가능한 에이전트 확장**이 목표다. 의존성 선택도 이 방향을 반영한다. 모델 실행 자체보다 `@kessler/gemma-agent`를 통한 agent integration과 브라우저 integration에 설계 무게중심이 있다. 즉, 이 repo의 핵심 문제는 "어떻게 추론하나"가 아니라 "어떻게 에이전트로 연결하나"다.

wxt.config.ts의 의미

단순 번들 설정이 아니다

이 파일은 런타임 배포 전략이 담겨 있다. 세 가지 핵심 결정이 여기에 있다.

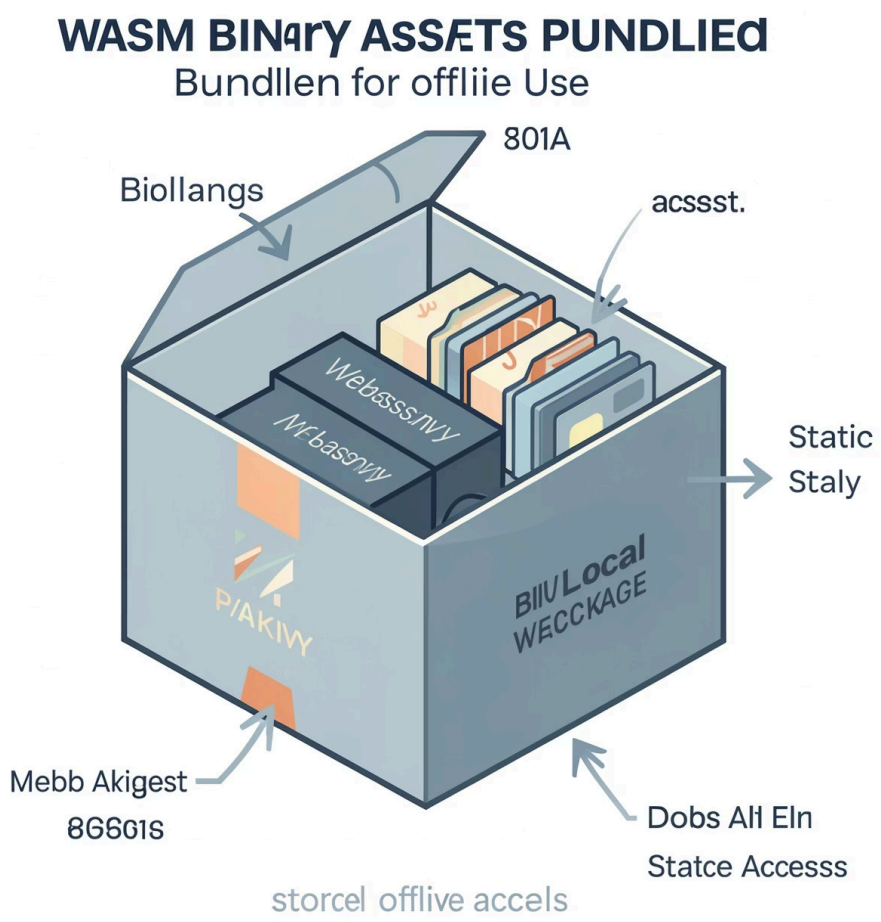
> **ORT 자산 복사**
>
> `onnxruntime-web` 자산을 빌드 시 `public/ort`로 복사해 로컬 패키징

> **CSP 설정**
>
> `wasm-unsafe-eval` 허용으로 추론 런타임 실행 가능하게 함

> **권한 선언**
>
> permissions와 host_permissions가 에이전트 기능 범위를 결정

⚠ wxt.config.ts를 변경하면 확장의 실행 가능 범위와 보안 표면이 직접 바뀐다. 이 파일은 아키텍처 결정이 코드화된 곳이다.

ONNX Runtime Web 자산 복사의 이유



브라우저에서 Gemma 4 ONNX 모델을 실행하려면 `onnxruntime-web`의 wasm/mjs 런타임 자산이 반드시 필요하다. 이를 CDN에서 동적으로 불러오는 대신 **확장 패키지 내 로컬 자산으로 포함**시키는 전략이다.

CDN 의존성 제거

외부 네트워크 없이도 런타임 로딩

오프라인 동작

인터넷 없어도 추론 실행 가능

실행 일관성

버전 고정으로 예측 가능한 동작

CSP와 추론 런타임

`wasm-unsafe-eval` 허용의 의미

Content Security Policy에서 `wasm-unsafe-eval`을 허용하는 것은 브라우저 ML 런타임 실행과 직결된다. 일부 wasm 기반 동작은 이 설정 없이 실행이 제한된다.

> ⓘ 이 설정은 단순 UI 확장과 추론 런타임 내장 확장을 구분 짓는 경계선이다. Gemma-gem은 명확히 후자를 선택했다.

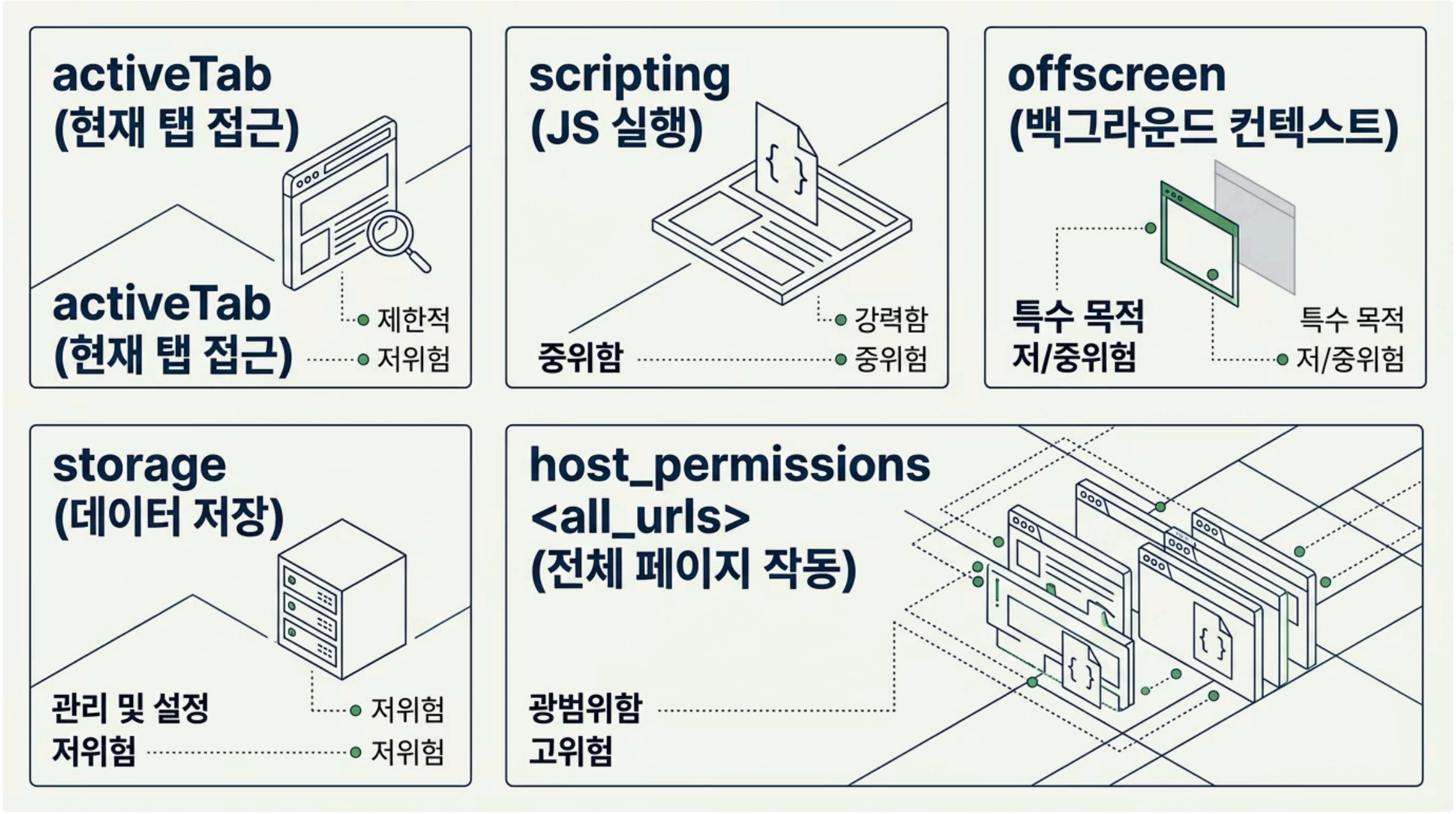
없으면

wasm 기반 ONNX 런타임 동작 제약, 모델 추론 불가

있으면

WebGPU + wasm 백엔드로 Gemma 4 전체 추론 파이프라인 가동

권한 설계



권한은 기능 범위이자 보안 표면이다. `activeTab`·`scripting`·`offscreen`·`storage`는 에이전트 동작에 필수적이며, `host_permissions: <all_urls>`는 거의 모든 페이지에서의 작동을 가능하게 한다. 기능은 강력하지만 그만큼 통제 책임도 커진다.

background.ts의 역할

Service Worker 진입점

확장 시작 시 두 가지 핵심 작업을 수행한다:

1. message router 설치 — 전체 메시지 버스 초기화
2. offscreen document 선제적 준비 — 첫 대화 전 모델 컨텍스트 미리 생성

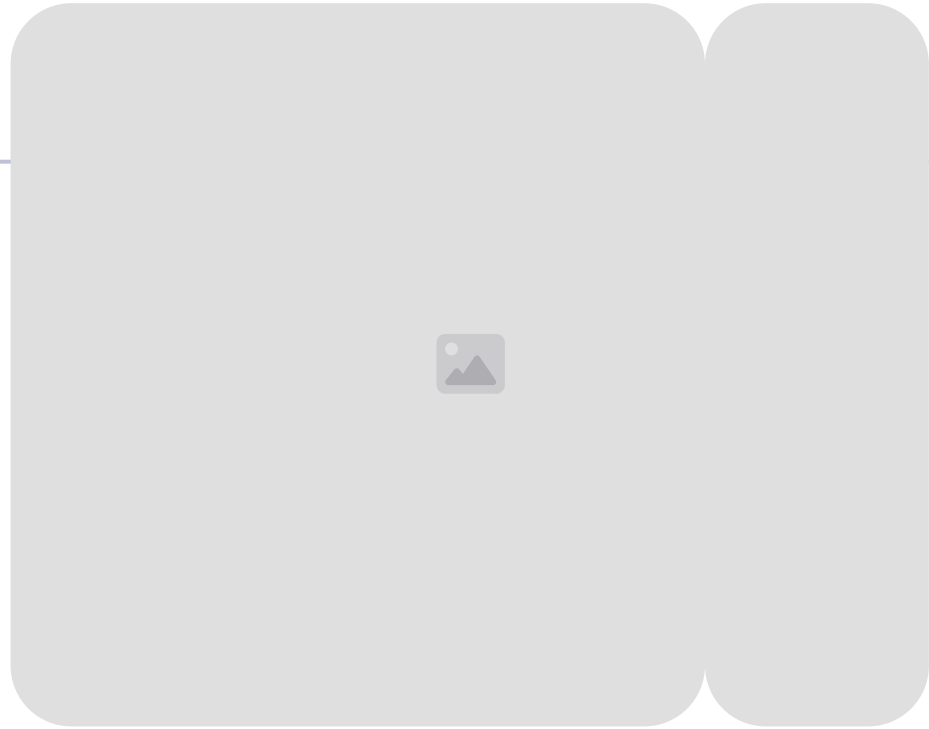
의도는 명확하다. 온디바이스 모델은 초기 로딩 비용이 크다. 사용자가 첫 메시지를 입력하는 시점이 아니라 **확장이 시작되는 시점**에 offscreen 준비를 시작해 체감 응답 시간을 낮춘다.

이것은 작은 코드 선택이지만 UX에 직접적인 영향을 주는 선제적 latency 완화 전략이다.

왜 offscreen을 미리 띄우는가

프리로드 없음

사용자 전송 → 오프스크린 생성 → 모델 로드 → 지연 발생



프리로드 있음

확장 시작 → 오프스크린 준비 → 즉시 추론 시작

온디바이스 모델의 초기 로딩은 모델 파일 파싱, 가중치 로딩, WebGPU 컴파일 등 수 초에서 수십 초가 걸릴 수 있다. 첫 상호작용 시점에 이 비용이 몰리면 **체감 성능이 급격히 나빠진다**. 선제적 offscreen 준비는 구현 한 줄의 변화지만 사용자 경험에서는 결정적 차이를 만든다.

offscreen-manager.ts 해설

핵심 기능

1 존재 여부 확인

offscreen document가 이미 생성됐는지 먼저 확인한다.

2 중복 생성 방지

`creating Promise`를 공유해 race condition을 제어한다.

3 단일 인스턴스 보장

복수 이벤트가 동시에 생성을 트리거해도 하나만 생성된다.

> Service worker는 재시작될 수 있다. 재시작 후 기존 offscreen이 살아 있을 수 있으므로, 이 manager는 중복 생성 경쟁을 막는 최소한의 lifecycle guard 역할을 한다.

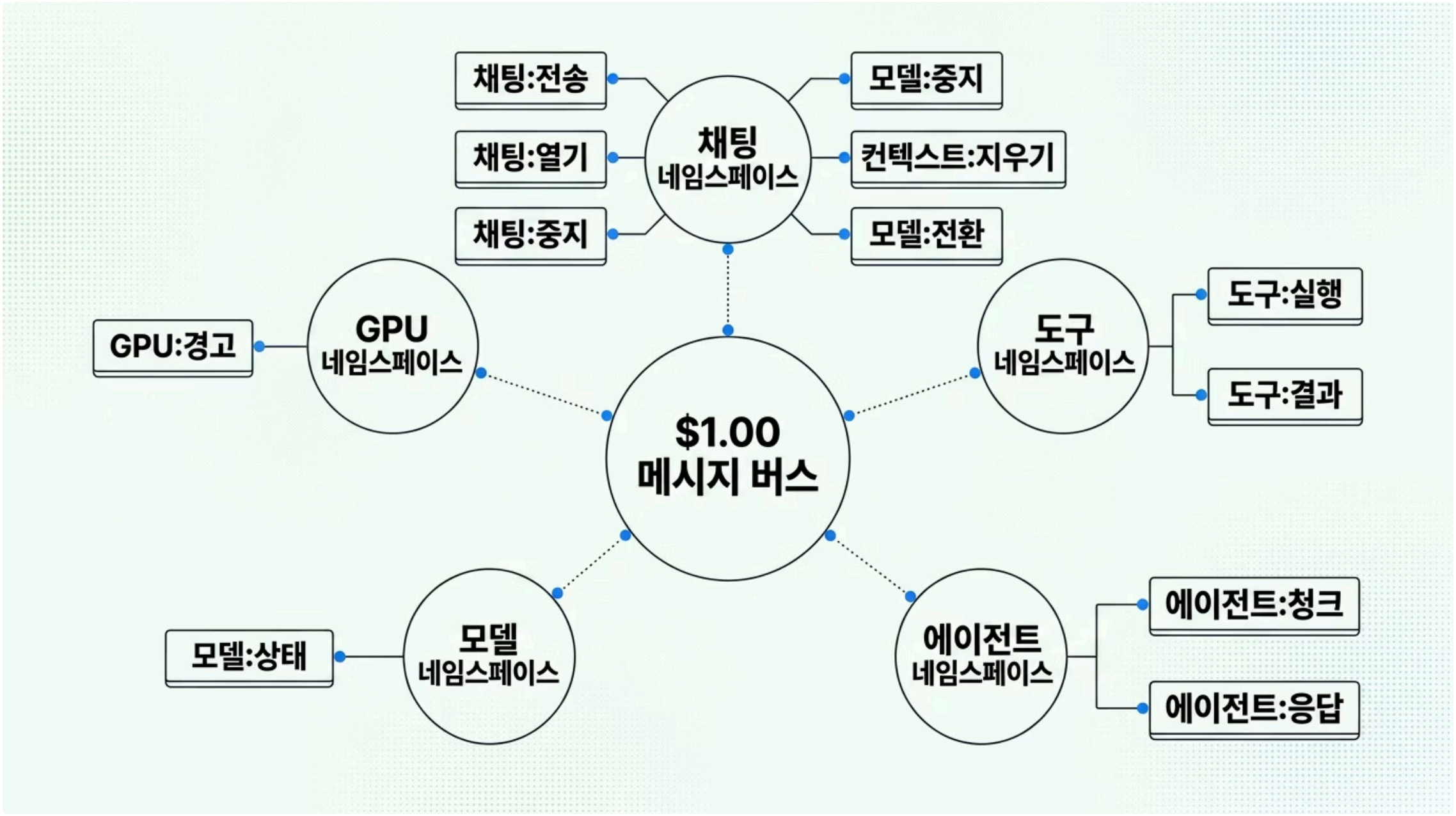
```
// 핵심 패턴
if (existingContexts.length > 0) return;
if (creating) { await creating; return; }
creating = chrome.offscreen.createDocument(...);
```

offscreen-manager 설계의 의미

Service worker는 Chrome이 필요하다고 판단하면 언제든 종료하고 재시작한다. 이 재시작 사이클은 예측하기 어렵다. 따라서 offscreen document의 lifecycle을 별도로 관리하지 않으면 모델 런타임이 불안정해진다.

문제	해결	결과
SW 재시작 → offscreen 중복 생성 race condition 발생 가능	creating Promise 공유로 최소한의 concurrency control 구현	복잡한 state machine 없이 안정적 단일 offscreen 인스턴스 유지

message-router.ts는 사실상 시스템 버스



모든 실행 컨텍스트는 이 메시지 계약 위에서 연결된다. 각 메시지 타입은 네임스페이스로 구분되어 있으며, 어느 컨텍스트가 발신하고 어느 쪽이 수신하는지가 아키텍처 경계를 정의한다.

메시지 기반 구조의 장점

컨텍스트 독립성

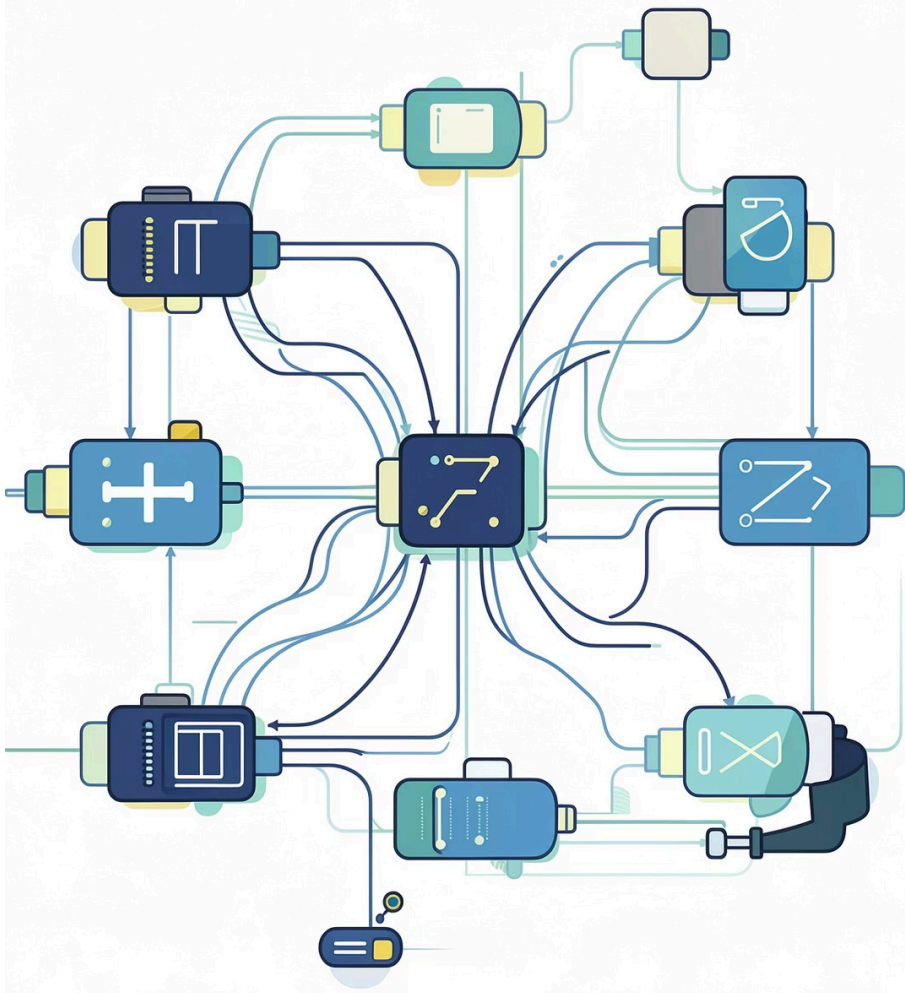
각 실행 컨텍스트가 상대방의 내부 구현을 알 필요가 없다. 메시지 계약만 지키면 된다.

프로세스 분리 수용

확장 환경의 프로세스 분리를 억지로 극복하려 하지 않고 자연스럽게 구조로 받아들인다.

테스트와 교체 용이

메시지 인터페이스만 맞추면 특정 컴포넌트를 독립적으로 교체하거나 모킹할 수 있다.



메시지 기반 구조의 단점

추적 복잡도 증가

이벤트 종류가 늘수록 어떤 메시지가 어디서 어디로 가는지 파악하기 어려워진다.

디버깅 방식 변화

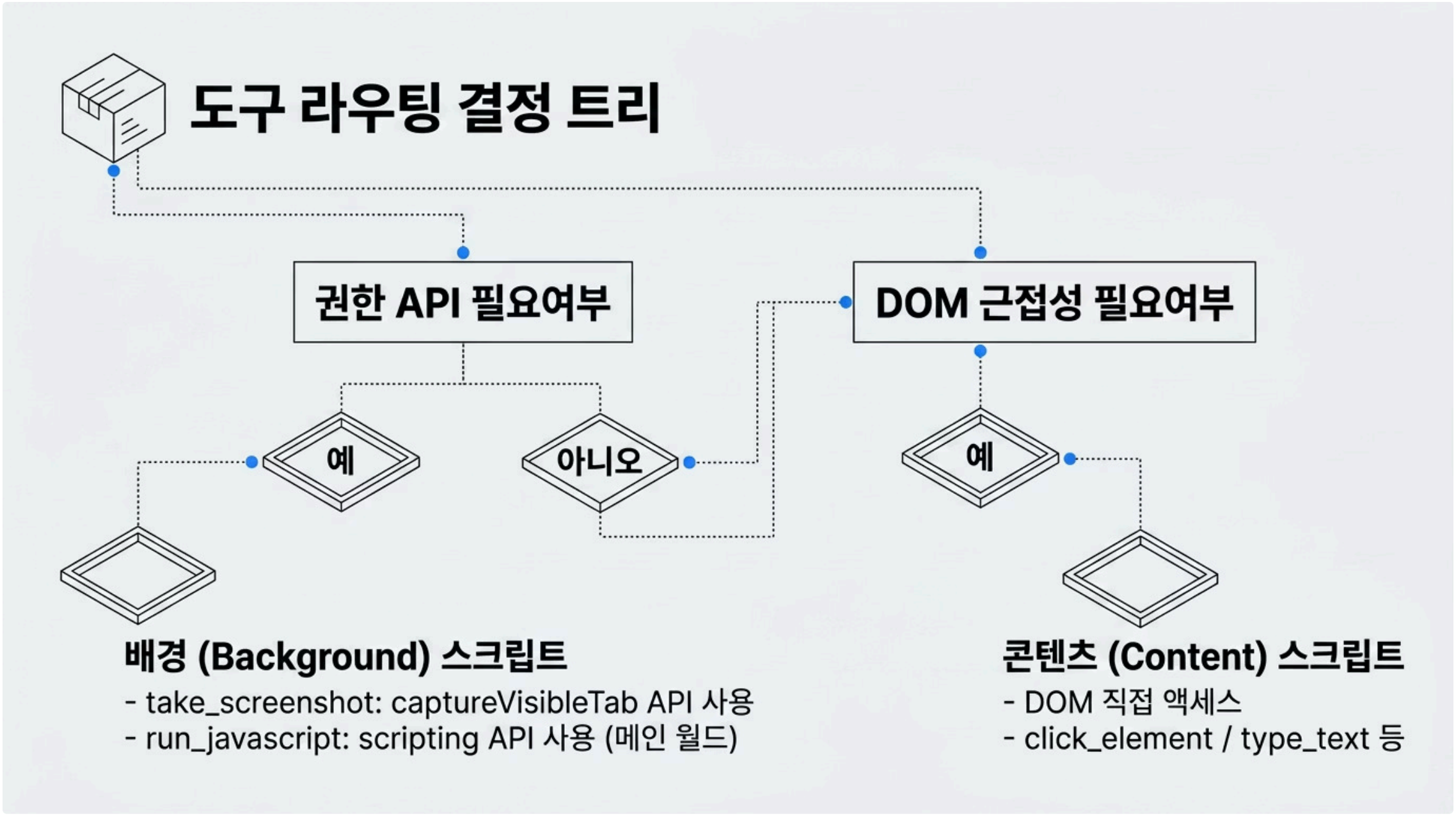
call stack 추적이 아니라 비동기 event chain 추적으로 디버깅 방식이 바뀐다.

대규모화 시 필수 인프라

로깅, tracing, correlation ID가 없으면 사고 추적이 매우 어려워진다.

⚠ 현재 저장소 규모에서는 관리 가능하지만, 메시지 타입이 20개를 넘어가는 시점부터 별도 메시지 추적 레이어가 필요해진다. 이것은 메시지 기반 아키텍처의 보편적 트레이드오프다.

tool execution 라우팅 전략



모든 도구가 같은 위치에서 실행되지 않는다. **privileged API 필요 여부**와 **DOM 근접성**을 기준으로 background와 content script로 나눈 이중 라우팅 전략이다.

take_screenshot 구현 의미

`chrome.tabs.captureVisibleTab`은 background 쪽 privileged API로, content script에서는 직접 호출할 수 없다. 이 도구를 통해 **모델이 페이지의 시각적 상태를 보완적으로 파악**할 수 있다.

텍스트 snapshot의 한계 보완

DOM 텍스트만으로는 캡처할 수 없는 레이아웃·차트·이미지 정보를 시각적으로 전달

멀티모달 컨텍스트

Gemma 4의 멀티모달 능력을 활용할 수 있는 입력 경로를 제공

run_javascript 구현 의미

⚠️ 가장 강력하고 가장 위험한 도구

page main world에서 임의의 JavaScript를 직접 실행한다. 이는 CSS selector 기반 도구로 처리하기 어려운 어떤 작업도 처리할 수 있음을 의미한다.

장점	위험
복잡한 페이지 상태 조작, framework-controlled 요소 접근, 동적 데이터 읽기 가능	모델이 생성한 임의 코드를 실행하는 구조. policy layer가 없으면 사고 반경이 매우 크다.

⊗ 로컬 실험용 에이전트에는 강력한 도구지만, 제품 수준 안전장치 없이는 사용에 주의가 필요하다.

run_javascript가 중요한 이유

브라우저 에이전트가 CSS selector만으로 작동하려 할 때 가장 자주 실패하는 상황이 있다. 동적으로 렌더링되는 컴포넌트, Shadow DOM 내부 요소, framework가 event를 가로채는 입력 필드 등이다. `run_javascript`는 이런 상황을 우회하는 마지막 수단이다.

우회 가능한 상황

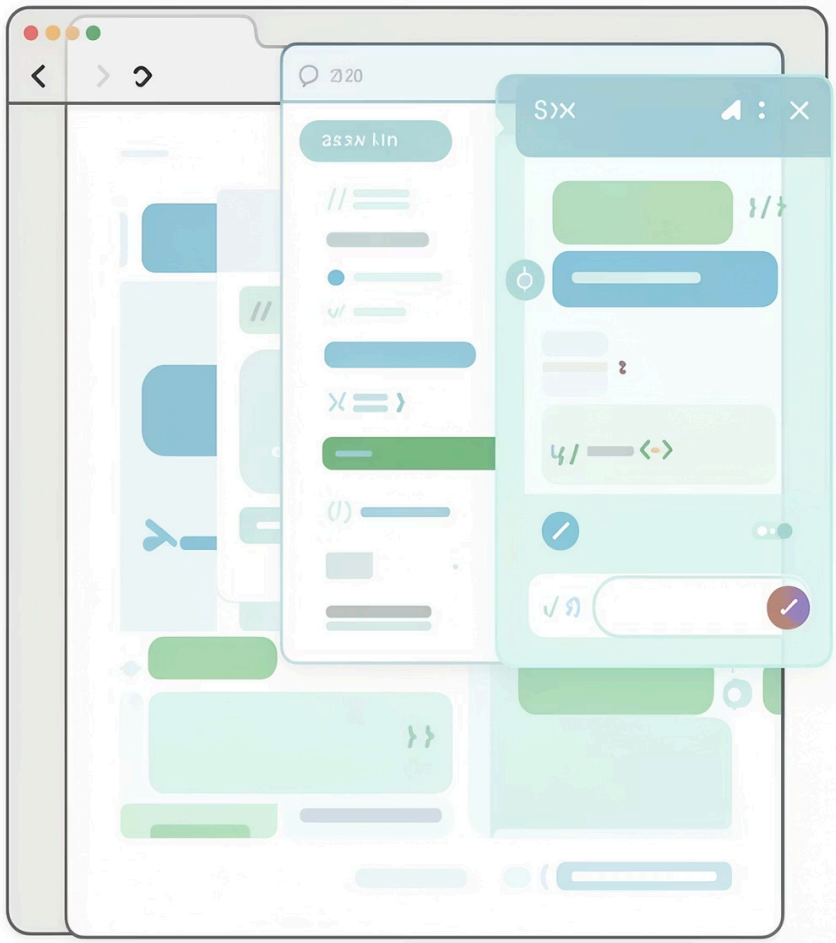
CSS selector로 접근 불가한 동적 요소, 복잡한 SPA 상태 읽기, 커스텀 이벤트 발화

위험 시나리오

모델이 잘못된 코드를 생성하면 페이지 데이터 유출, 리다이렉션, 의도치 않은 폼 제출 발생 가능

현재 상태

policy layer, sandboxing, user approval 없이 직접 실행. 실험용 에이전트로의 포지션 확인 필요



content.ts 개요

페이지 위 UI 삽입

각 페이지마다 gem 아이콘과 채팅 오버레이를 삽입해 사용자 진입점을 만든다.

메시지 브리지

사용자 입력을 받아 background로 전달하고, 스트리밍 응답을 UI에 반영한다.

DOM 도구 실행

background에서 위임된 DOM 중심 tool call을 실행하고 결과를 반환한다.

페이지 내 채팅 UI의 역할

> Gemma-gem은 독립 앱이 아니라 **현재 페이지 위에서** 작동한다. 사용자는 탭을 떠나지 않고 즉시 질문하고 조작을 요청할 수 있다.

이 설계는 단순한 편의 기능이 아니다. 페이지와 모델을 **동일한 맥락**에 놓는다는 의미다. URL, 페이지 제목, 현재 보이는 콘텐츠가 자연스럽게 대화 컨텍스트에 포함된다.

컨텍스트 유지

탭 전환 없이 현재 페이지 맥락 유지

즉각적 상호작용

보고 있는 것을 바로 질문하거나 조작 요청

신뢰 형성

결과가 페이지 위에서 즉시 확인 가능

page snapshot 전략

메시지 전송 시 URL, title, body text 일부를 잘라 `pageContext`로 함께 전송한다. 최대 길이를 제한해 토큰과 latency를 통제한다. 모든 질문에서 도구를 먼저 호출하지 않고 **값싼 정적 컨텍스트를 우선 활용**하는 전략이다.

왜 snapshot-first 전략이 좋은가

핵심 원칙

에이전트가 무조건 도구를 먼저 호출하는 것은 낭비다. 정적 컨텍스트로 즉시 답할 수 있는 질문은 추가 tool call 없이 처리한다.

→ **inference cost 절감**

불필요한 도구 호출 왕복을 줄여 전체 추론 비용 감소

→ **interaction latency 단축**

tool call RTT 없이 즉시 응답 가능한 경우 처리

→ **agentic loop 과부하 방지**

모든 것을 heavy automation으로 만들지 않는 균형 잡힌 설계

UI 스트리밍 처리

Thinking chunk

모델이 내부적으로 추론 중인 상태를 별도 표시. 사용자가 시스템이 작동 중임을 시각적으로 확인.

Tool chunk

도구가 호출됐거나 결과를 기다리는 상태를 구분 표시. 어떤 도구가 실행 중인지 맥락 제공.

텍스트 chunk

실제 최종 답변이 스트리밍으로 출력됨. 긴 응답도 첫 토큰부터 즉시 표시.

단순히 최종 응답만 보여주는 것이 아니라 내부 상태를 투명하게 노출해 사용자 신뢰와 UX를 동시에 높인다.

이런 스트리밍 UX의 기술적 의미

에이전트는 응답 시간이 가변적이다. tool call이 하나 추가될 때마다 수 초가 더 걸릴 수 있다. 이 동안 화면이 멈춘 것처럼 보이면 사용자는 오작동으로 인식한다.

perceived latency 감소

부분 상태 노출로 체감 대기 시간이 크게 줄어든다

디버깅 가능성 향상

개발자가 어느 단계에서 지연이 생기는지 즉시 파악 가능

신뢰 형성

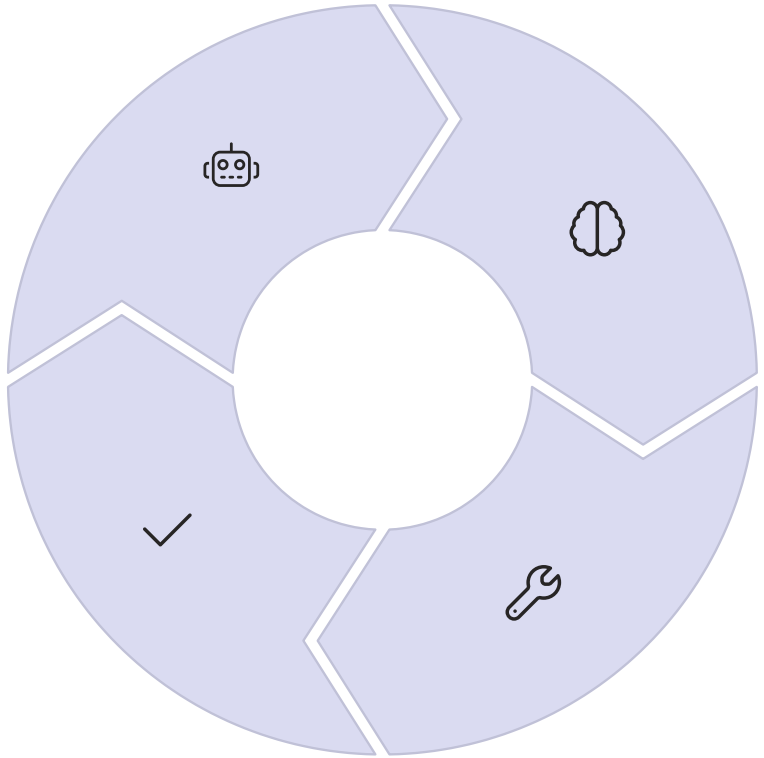
내부 동작의 투명성이 사용자 신뢰를 높인다

offscreen/main.ts 개요

> 실제 agent orchestration이 일어나는 중심 파일. 모델, 도구, 스트리밍, 상태 관리가 모두 여기에 연결된다.

Agent 생성

tool definition 연결 포함



모델 추론 요청

스트리밍으로 chunk 수신

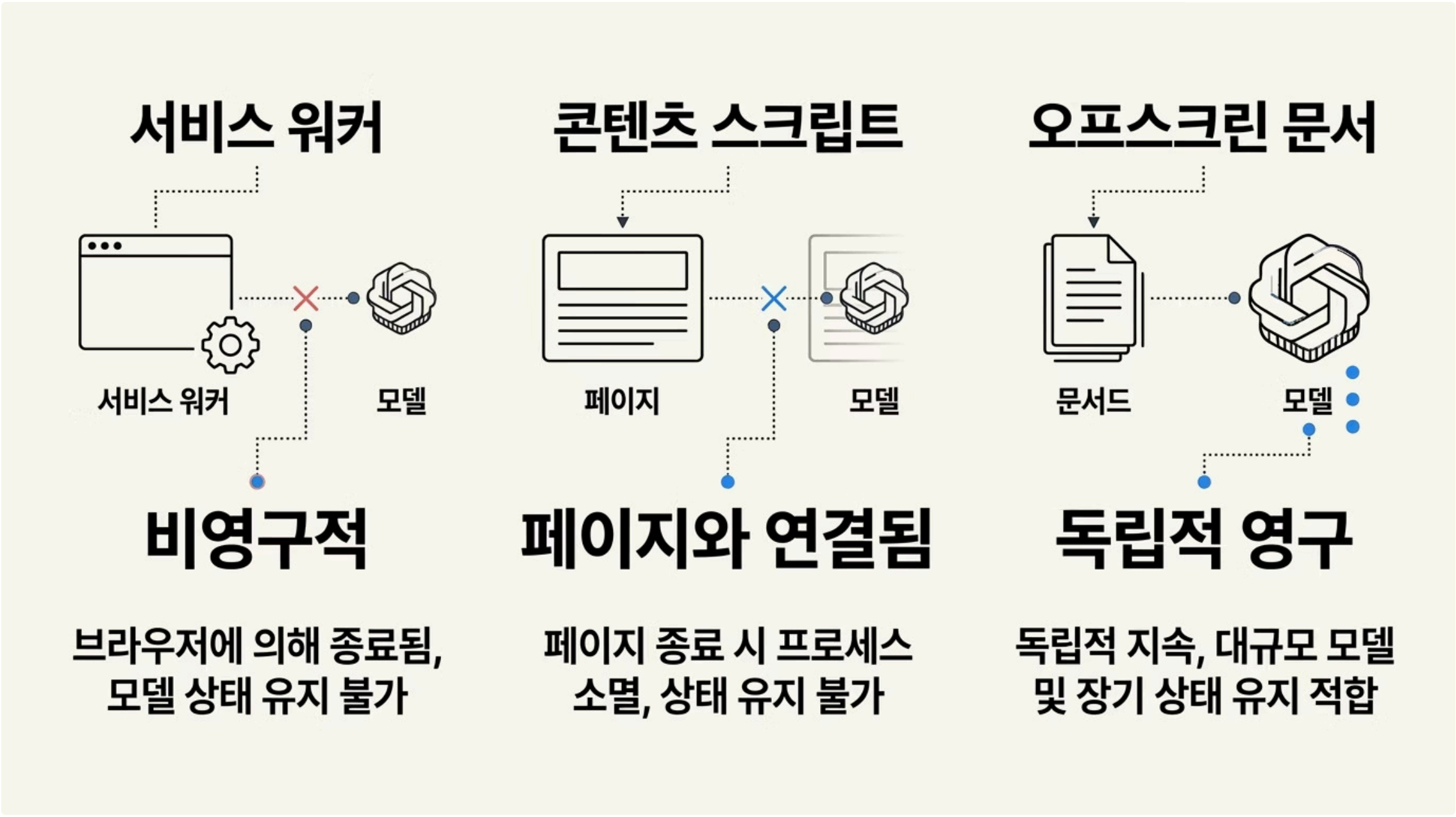
Tool call 발화

requestId 생성 후 background로 전달

결과 수신 및 재개

pending map resolve 후 계속 진행

왜 offscreen document가 핵심인가



offscreen document는 이 repo에서 사실상 **모델 프로세스 역할**을 한다. 모델 가중치 로딩, 추론 실행, agent 상태 유지 모두 여기서 담당한다. 다른 컨텍스트에서는 이 역할을 안정적으로 수행하기 어렵다.

Agent와 Tool의 연결 방식

선언적 TOOL_DEFINITIONS

에이전트에게 어떤 도구가 있고 어떤 파라미터를 받는지 선언한다. 실제 실행 코드는 포함하지 않는다.

메시지 기반 RPC 실행

offscreen은 도구를 직접 실행하지 않는다. `requestId`를 생성해 `tool:execute` 메시지로 background에 위임한다.

1

offscreen

tool:execute + requestId 발신

2

background

라우팅 후 실행 위임

3

tool:result

pending map에서 requestId로 resolve

이 RPC 구조의 장점

관심사 분리

에이전트는 브라우저 API와 DOM 세부 구현을 전혀 몰라도 된다. tool:execute 메시지만 보낼 뿐이다.

교체 용이성

tool 실행 위치를 나중에 변경해도 offscreen의 계약은 유지된다. 메시지 타입만 지키면 된다.

비동기 표준화

확장 환경에서 동기 호출 대신 메시지 기반 비동기 호출을 일관된 패턴으로 표준화한다.

pendingToolResults와 timeout

왜 필요한가

tool 결과가 영원히 오지 않으면 agent loop 전체가 멈춘다. 브라우저 에이전트는 DOM 조작, 스크린샷, JS 실행 등 외부 세계와 상호작용이 많아 실패 격리가 필수다.

```
// 개념적 패턴
const pendingToolResults = new Map();
// tool:execute 발신 시
pendingToolResults.set(requestId, {
  resolve, reject,
  timeout: setTimeout(() => reject('timeout'), 5000)
});
```

⚠ 이 설계는 작지만 매우 중요하다. timeout 없이는 하나의 도구 실패가 전체 에이전트 세션을 영구적으로 멈추게 만들 수 있다.

requestId 발급

각 tool call마다 고유 ID

timeout 설정

무한 대기 방지

결과 매핑

ID로 결과 resolve

chat:stop과 abort 흐름

사용자가 중단을 요청하면 모델 생성과 agent 실행을 `AbortController`로 중단한다. 긴 추론이나 잘못된 tool loop를 끊을 수 있게 한다. 단, **이미 실행된 페이지 조작은 되돌리지 않는다**. cancellation은 computation 중심이지 side effect rollback이 아니다.

시스템 프롬프트 설계

포함되는 정보

현재 날짜·시간 및 locale 기반 국가 정보

모델이 시간 맥락을 가지고 응답하게 한다

snapshot-first 지시

page snapshot이 있으면 먼저 그것으로 답하도록 유도

도구 사용 제한

현재 페이지와 직접 연결된 도구만 사용하라는 명시적 제약

프롬프트 전체 톤은 간결하고 운영 지향적이다. 긴 설명보다 행동 지침에 집중한다. 이 설계는 불필요한 일반 대화를 줄이고 페이지 맥락에 집중하게 만드는 효과를 낸다.

프롬프트 설계의 장점과 한계

✅ 장점

→ 페이지 맥락 집중 유도

불필요한 범용 대화 대신 현재 페이지 작업에 집중하게 만든다

→ 도구 최소화 유도

snapshot-first로 tool call 횟수를 줄여 비용·속도 개선

⚠️ 한계

→ 세밀한 지침 부재

destructive action 승인, selector 실패 fallback, 탐색 전략에 대한 구체적 지침이 얇다

→ 모델 자체 추론 의존도 높음

엣지 케이스 처리를 모델의 자체 판단에 크게 위임한다

model-host.ts는 추론 런타임의 핵심

```
// 핵심 초기화 코드 (개념적)
const model = await Gemma4ForConditionalGeneration
  .from_pretrained(modelId, {
    device: "webgpu",
    dtype: "q4f16"
  });

const processor = await AutoProcessor
  .from_pretrained(modelId);

// ORT 경로를 확장 자산으로 연결
env.backends.onnx.wasm.wasmPaths =
  chrome.runtime.getURL("ort/");
```

device: webgpu

GPU 가속 실행 경로

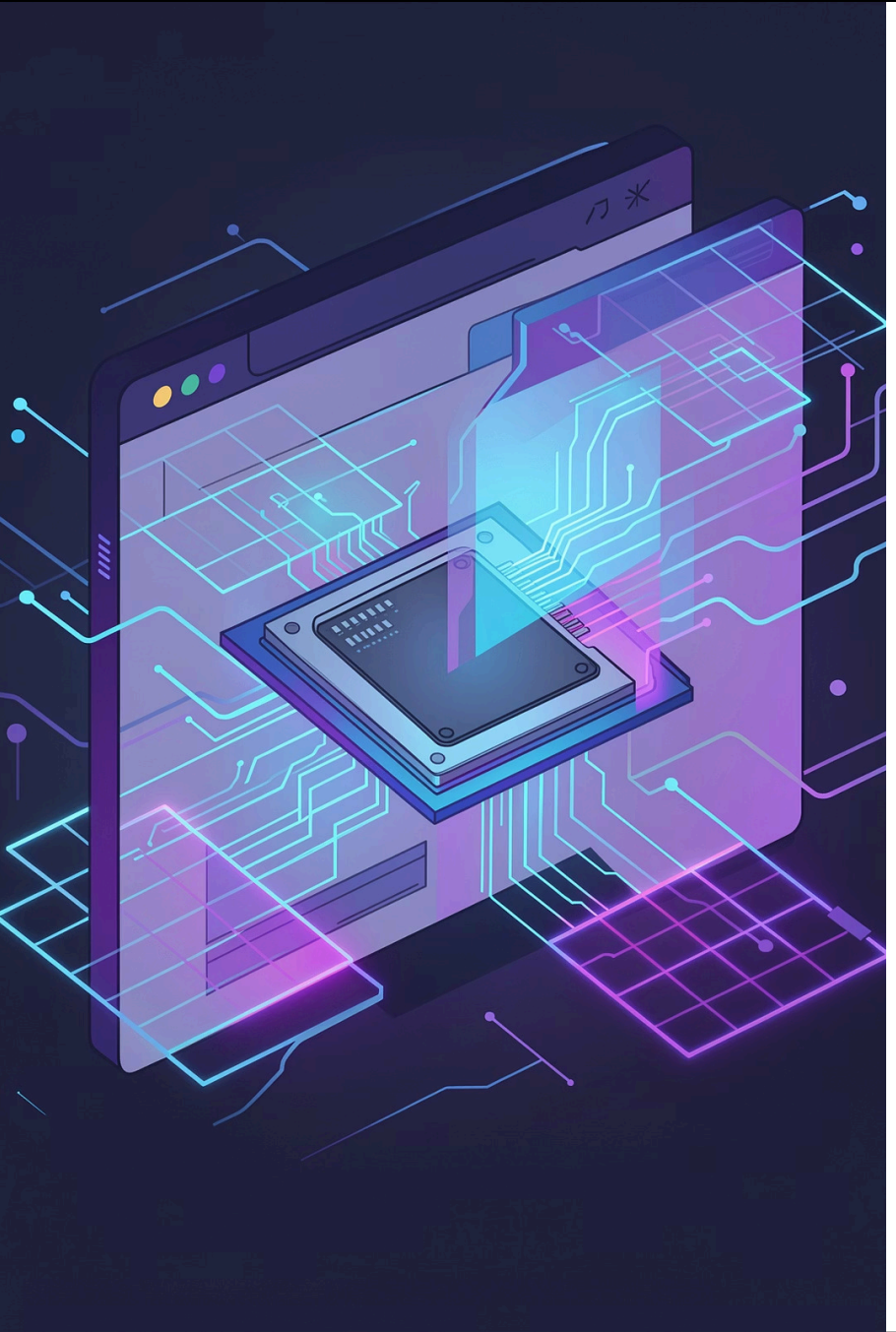
dtype: q4f16

4-bit 양자화로 메모리 최적화

ORT path 연결

로컬 자산으로 런타임 해결

이 세 줄의 설정이 브라우저 내 Gemma 4 추론을 가능하게 하는 핵심이다.



왜 WebGPU가 중요한가

CPU 한계

CPU만으로는 브라우저 내 대형 모델 추론이 너무 느려 실용적이지 않다. 수십 초 이상의 응답 시간은 사용 불가 수준이다.

WebGPU 가속

브라우저에서 GPU 컴퓨팅을 활용하는 표준 API. 행렬 연산 병렬화로 추론 속도를 수십 배 향상시킨다.

Gemma-gem의 전제

WebGPU 없이는 이 UX 자체가 성립하지 않는다. 모든 사용성 설계가 GPU 가속을 기반으로 한다.

모델 선택 구조

Gemma 4 E2B

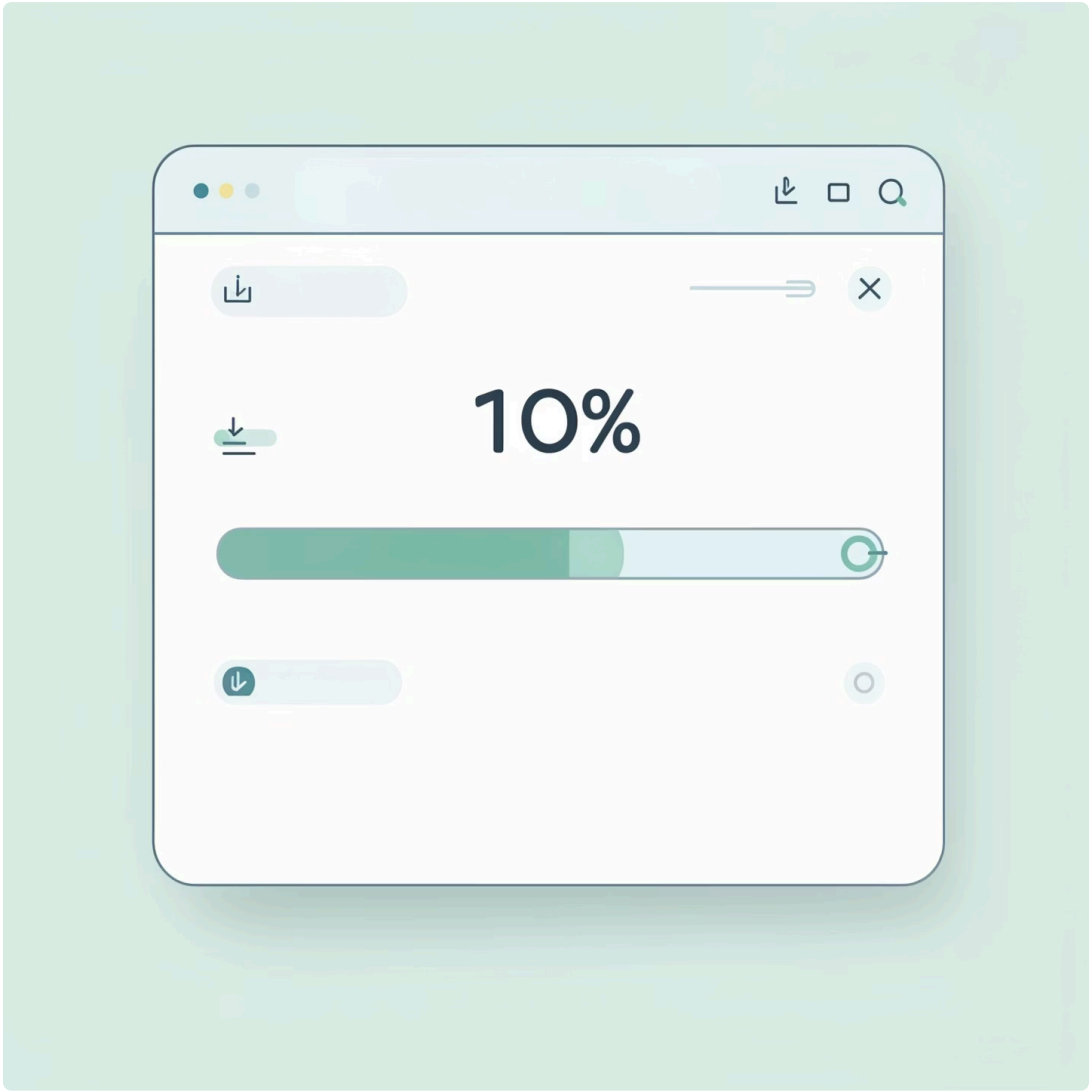
더 작은 모델. 다운로드 크기와 GPU 메모리 사용량이 낮다. 성능은 낮지만 하드웨어 여건이 제한적인 환경에 적합하다.

Gemma 4 E4B

더 큰 모델. 추론 품질이 높다. 충분한 GPU 메모리와 더 나은 성능을 원하는 사용자에게 적합하다.

두 옵션을 제공함으로써 저사양 기기부터 고사양 기기까지 다양한 하드웨어 여건을 수용한다. 저장소는 범용 모델 허브가 아니라 **Gemma 4 계열에 집중**하는 것이 특징이다. 모델 선택 기능 자체가 `model:switch` 메시지를 통해 런타임에 처리된다.

progress_callback의 역할



모델 다운로드와 로딩은 최초 실행 시 수 GB에 달하는 데이터 처리가 필요하다. `progress_callback`은 이 과정을 수집해 사용자에게 퍼센트로 보여준다.

바이트 정확도보다 체감 피드백

정확한 수치보다 진행 중이라는 신호 자체가 중요하다

모델 상태 메시지 연동

model:status 메시지로 UI에 실시간 전달

UX 결정적 요소

진행률 없는 로딩은 사용자가 오작동으로 오해하게 만든다

TextStreamer 기반 스트리밍

`TextStreamer`는 모델 생성 중 토큰을 순차적으로 콜백으로 전달하는 인터페이스다. 전체 생성이 끝나길 기다리지 않고 즉시 chunk 단위로 UI에 전송할 수 있다.

즉각적 첫 토큰

사용자는 생성 시작과 동시에 첫 글자부터 볼 수 있다. 체감 응답 시간이 크게 줄어든다.

장문 응답 처리

긴 응답이나 도구를 여러 번 호출하는 작업에서 시스템이 계속 동작 중임을 알 수 있다.

agent:chunk 이벤트

각 토큰 chunk가 agent:chunk 메시지로 content script UI로 전달된다.

Gemma 출력 채널 파싱

이 repo는 단순 문자열 출력을 그대로 보여주지 않는다. Gemma 4의 특수 토큰과 태그를 파싱해 thinking·tool_call·tool_response·일반 텍스트로 분리한다. 모델 출력 형식을 **agent protocol처럼 재구성**하는 브리지 역할이다.

이 출력 파싱 방식의 의미

장점: 빠른 agent화

모델 출력 포맷을 파싱함으로써 structured API 없이도 tool-using agent를 빠르게 구현할 수 있다. 구현 속도와 유연성이 높다.

⚠️ 주의: 포맷 결합도

→ **모델 고유 토큰에 강하게 결합**

Gemma 4의 특정 출력 포맷에 의존하는 파싱 규칙

→ **모델 교체 시 깨질 위험**

동일한 parsing 규칙이 다른 모델 패밀리에 적용되지 않을 수 있음

→ **구조화 API 대비 취약성**

textual convention 의존도가 높아 모델 버전 업시 회귀 테스트 필요

content/tool-executors.ts 개요

read_page_content

selector와 반환 형식(text/html)을 받아 특정 영역의 내용을 읽어온다. 길이를 잘라 반환한다.

click_element

CSS selector로 요소를 찾고 클릭 이벤트를 발화한다. 버튼, 링크, 메뉴 등 상호작용 요소 조작.

type_text

input/textarea 요소에 value를 설정하고 input/change 이벤트를 발화한다.

scroll_page

페이지를 smooth behavior로 스크롤한다. lazy loading과 화면 밖 요소 탐색에 필요.

read_page_content 도구

동작 방식

CSS selector와 반환 형식 파라미터를 받아 해당 요소의 텍스트 또는 HTML을 읽어온다. 반환 길이를 제한해 토큰 오버플로를 방지한다.

```
// 개념적 구현
const el = document.querySelector(selector);
const content = format === 'html'
  ? el.innerHTML
  : el.textContent;
return content.slice(0, MAX_LENGTH);
```

page snapshot 보완

전체 페이지 스캔 대신 특정 영역을 정밀하게 읽어온다. 필요한 정보만 추출하는 목표 지향적 접근.

길이 제어

긴 콘텐츠를 잘라 모델 컨텍스트 윈도우와 추론 비용을 통제한다.

click_element 도구

CSS selector로 요소를 찾고 `.click()`을 호출한다. 구현은 단순하지만 실전에서는 **selector 품질이 성패를 좌우**한다. 이 도구가 신뢰할 만하게 작동하려면 모델이 페이지 DOM 구조를 정확히 추론해야 한다.

잘 동작하는 경우

명확한 ID나 class를 가진 단순 HTML 버튼, 링크, 체크박스. 정적으로 렌더링된 페이지.

어려운 경우

동적으로 생성된 요소, 중복 selector, Shadow DOM 내부, React/Vue synthetic event handler.

에이전트의 DOM 추론력이 중요한 이유가 여기 있다. 도구 자체보다 올바른 selector를 생성하는 모델의 능력이 병목이 된다.

type_text 도구

input/textarea 계열 요소에 value를 직접 설정하고 input/change 이벤트를 프로그래밍적으로 발화한다. 간단한 폼 입력에는 충분히 작동한다.

잘 동작

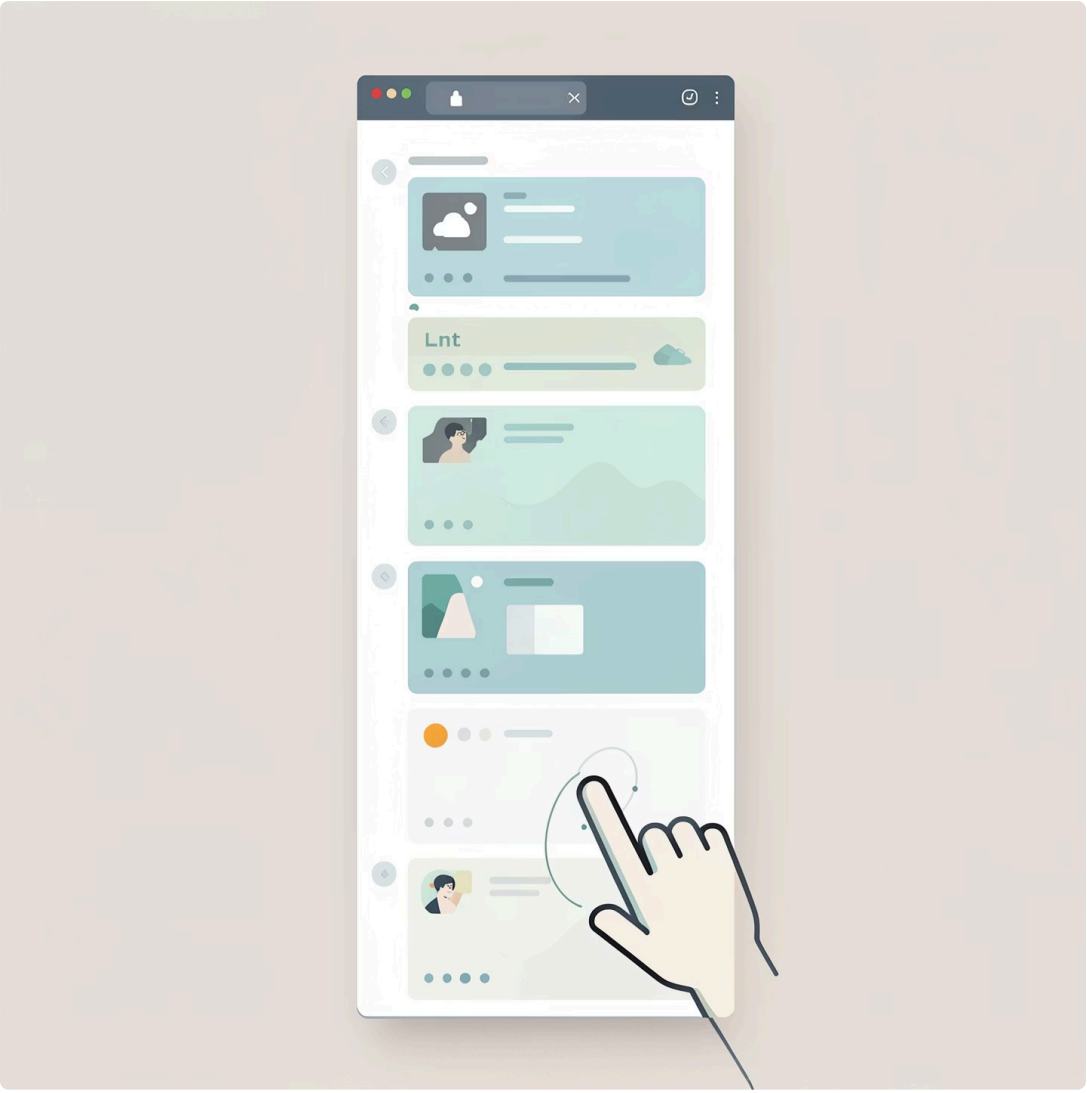
일반 HTML input, 전통적 jQuery 이벤트 핸들러, 단순 폼 필드

한계 존재

React controlled input (state로 관리), contenteditable 요소, 복잡한 rich text editor

⚠ framework-controlled input에서는 value 직접 설정이 React 상태와 불일치를 일으킬 수 있다. nativeInputValueSetter 같은 우회법이 필요한 경우가 있다.

scroll_page 도구



페이지를 `behavior: smooth`로 스크롤한다. 겉으로는 단순하지만 에이전트의 탐색 전략에서 중요한 역할을 한다.

→ **lazy loading 트리거**

스크롤로 새로운 콘텐츠가 동적 로드되는 페이지 탐색

→ **화면 밖 요소 접근**

viewport 밖의 요소를 가시 영역으로 가져와 다음 tool call에서 접근 가능하게 함

→ **시각 상태 변경**

스크린샷과 함께 사용 시 다른 시각적 상태 캡처 가능

현재 도구 세트의 성격

설계 철학

적은 수의 범용 도구로 다양한 작업을 커버하는 방식. 도구가 많을수록 에이전트의 선택 복잡도가 높아지고, 각 도구의 구현 유지 비용도 커진다.

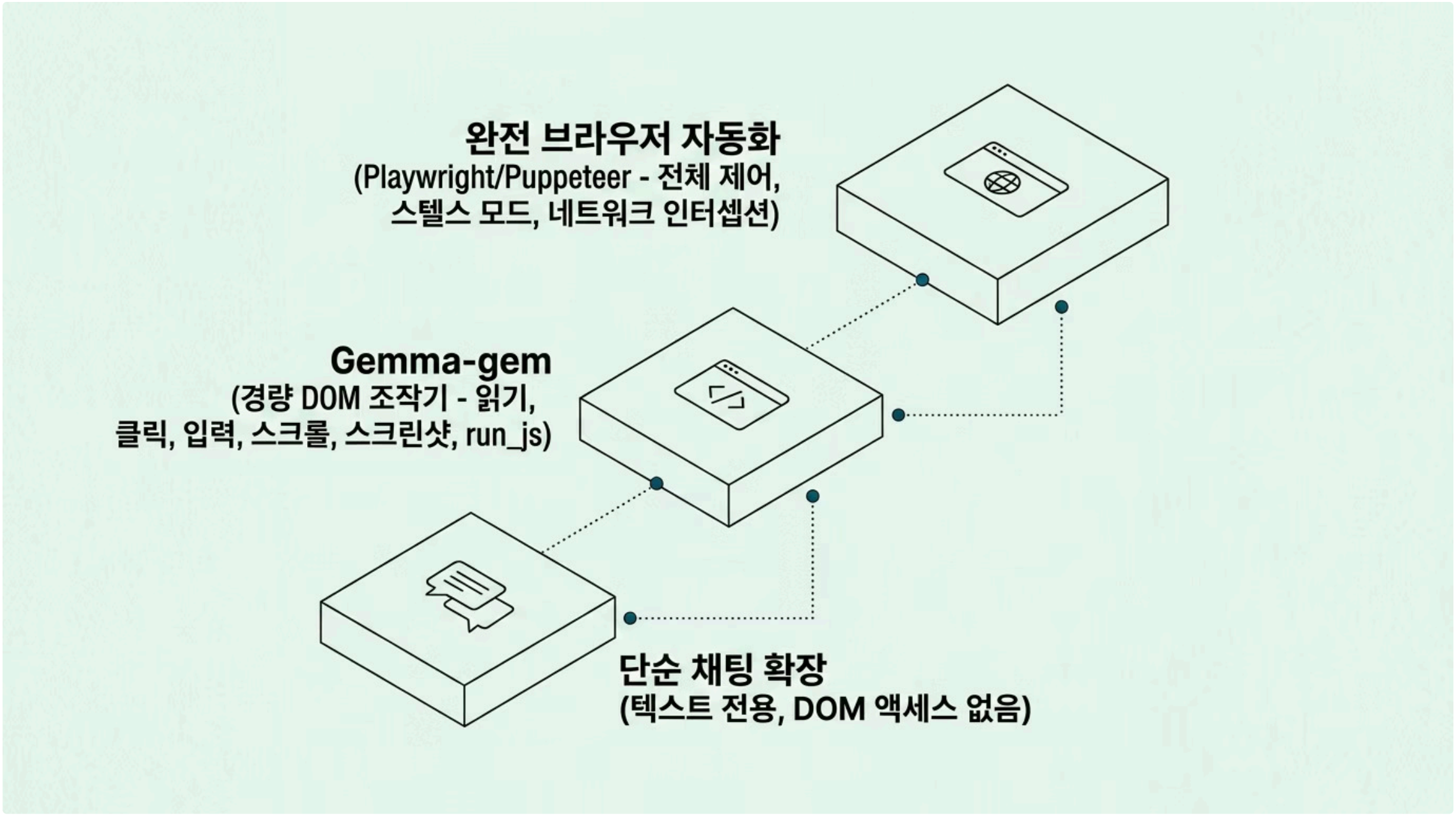
장점

구현이 작고 각 도구의 동작이 예측 가능. 디버깅이 쉽고 오버헤드가 낮다.

단점

복잡한 SPA 자동화에는 selector 실패나 이벤트 불일치로 quickly brittle해질 수 있다.

Gemma-gem의 자동화 수준은 어느 정도인가



Gemma-gem은 full browser automation framework가 아니다. **lightweight DOM operator**에 더 가깝다. 질의응답 + 가벼운 페이지 조작 + 필요 시 보조적 도구 호출이 현실적 포지션이다. 이 기대치를 명확히 이해해야 올바른 사용 판단을 할 수 있다.

shared/messages.ts의 가치

이 파일이 아키텍처의 실제 경계면

메시지 타입이 네임스페이스별로 정리된 이 파일은 단순한 상수 모음이 아니다. 확장 환경에서 **타입 계약은 곧 시스템 인터페이스**다. 이 파일을 읽으면 시스템이 무엇을 할 수 있는지 전체 그림이 나온다.

chat

send, open, stop, clear, switch

agent

chunk, response

tool

execute, result

model / gpu

status, warning

세션 모델

offscreen에는 사실상 `currentAgent`와 `currentTabId` 중심의 상태가 있다. 구조상 **한 번에 하나의 탭/세션**에 더 가까운 운영 모델이다. 복수 탭 병렬 에이전트 실행보다 단일 활성 세션이 우선된다.

model:switch나 context:clear 메시지가 오면 세션 상태가 리셋된다. 이 단일 세션 모델은 현재 구현의 의식적인 단순화 선택이다.

currentAgent

활성 에이전트 인스턴스. 대화 컨텍스트와 tool 연결 포함.

currentTabId

현재 활성 탭 식별자. 도구 실행 라우팅 기준.

abortController

현재 실행 중인 추론의 중단 제어 핸들.

이 단일 세션 모델의 장단점

장점 1: 단순한 상태 관리

tabId별 session map 없이 단일 상태만 관리하면 된다. 코드가 짧고 버그 가능성이 낮다.

장점 2: 모델 메모리 효율

하나의 모델 인스턴스를 공유하므로 GPU 메모리를 중복 사용하지 않는다.

단점: 확장성 한계

복수 탭을 동시에 다른 에이전트로 운영하는 시나리오를 지원하지 못한다. 미래 확장 시 tabId별 session map 도입이 필요하다.

UX 측면에서 중요한 점

모델 로딩 상태 노출

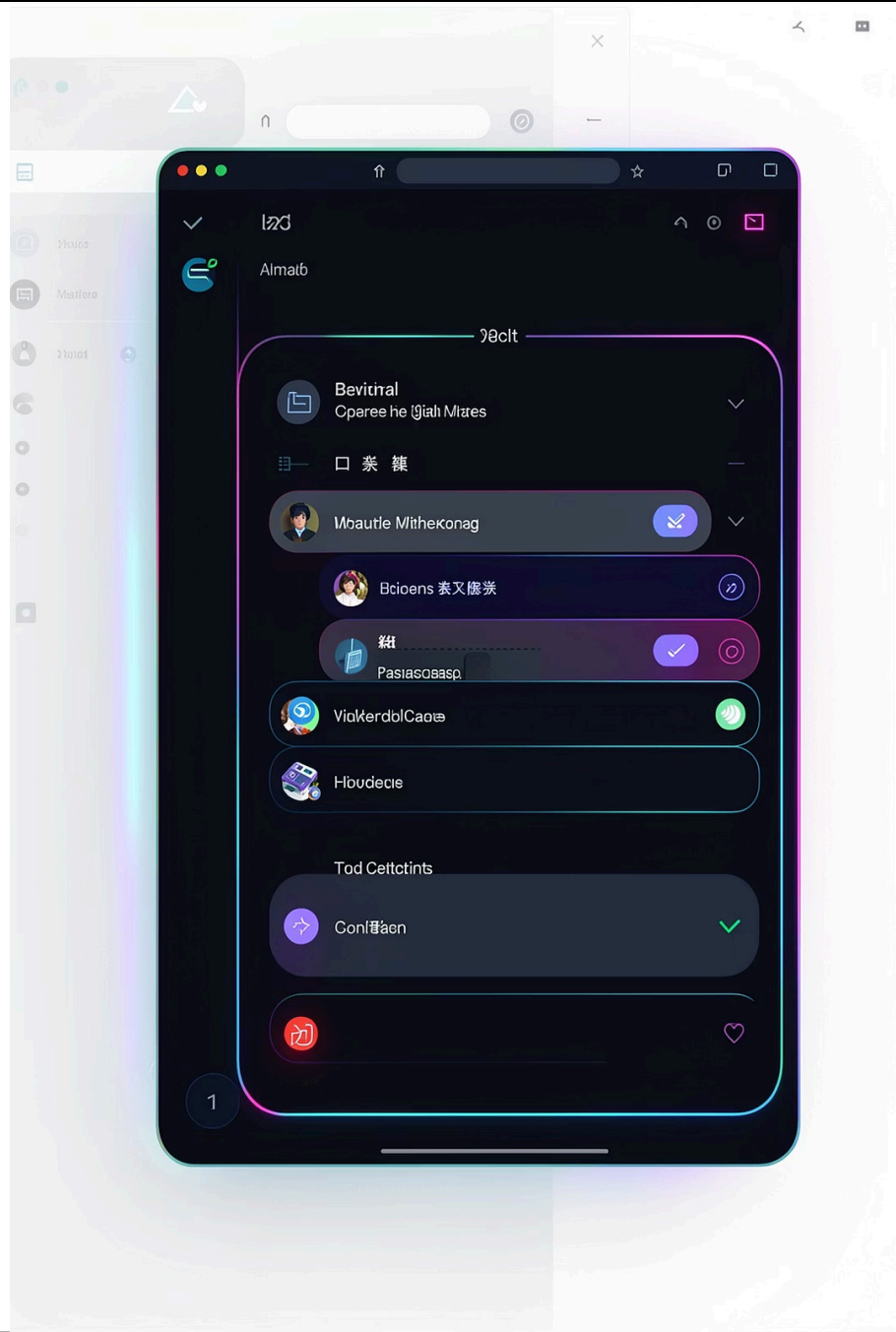
다운로드·로딩 진행률을 실시간으로 표시해 대기 중임을 명확히 알린다.

thinking/tool chunk 구분

모델 내부 추론과 도구 호출을 시각적으로 구분해 에이전트 동작을 투명하게 보여준다.

페이지 밀착 상호작용

탭을 떠나지 않고 현재 페이지 안에서 바로 질문하고 조작 결과를 확인한다.



프라이버시 관점

구조적 프라이버시 장점

→ 로컬 추론

페이지 내용이 외부 API로 자동 전송되지 않는 구조적 보장

→ 서버 의존성 없음

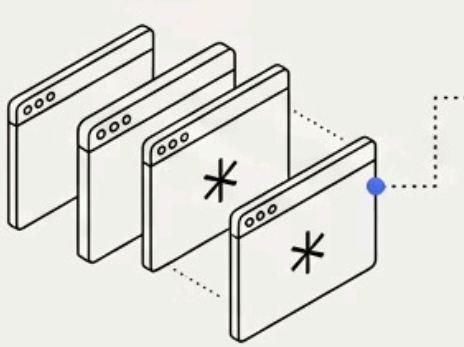
추론 결과가 서버에 기록되거나 학습에 사용되지 않음

⚠ **프라이버시 ≠ 안전성**

로컬 추론이라도 브라우저 권한과 run_javascript 때문에 로컬이라는 것만으로 무조건 안전하지 않다. 강력한 브라우저 조작 권한은 별도의 안전성 설계 문제다.

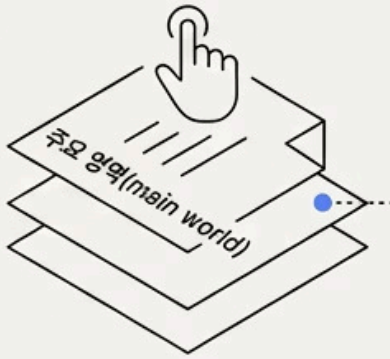
보안/안전성에서 가장 중요한 지점

1. host_permissions all_urls



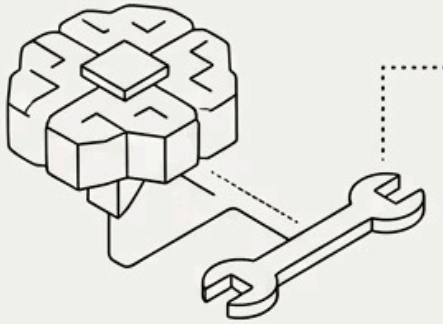
모든 URL 호스트 권한 (광범위한 공격 표면)

2. main world JavaScript execution



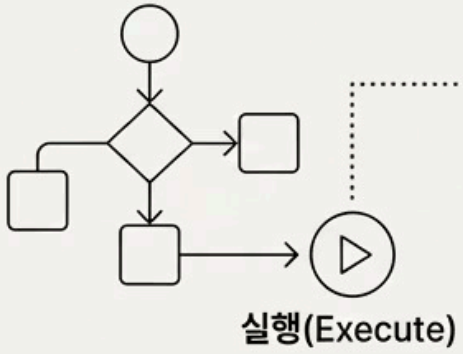
메인 월드 JavaScript 실행 (임의 코드 주입)

3. Model-generated tool calls



모델 생성 도구 호출 (승인 없는 파괴적 작업)

4. No policy layer



정책 계층 부재 (비가역적 작업 확인 없음)

기능성은 높지만 policy layer는 얇다. 이 네 가지 지점을 이해하지 않고 배포하면 의도치 않은 결과가 발생할 수 있다. 로컬 실험용으로 설계된 현재 구조에서 제품화로 이어지려면 각 지점에 대한 명시적 설계가 필요하다.

성능 병목 관점

1

Tool call RTT 누적

도구 호출이 많아질수록 각 왕복 지연이 쌓여 전체 응답 시간이 길어진다

2

GPU 메모리와 WebGPU 지원

브라우저의 WebGPU 지원 여부와 가용 GPU 메모리에 민감하게 반응한다

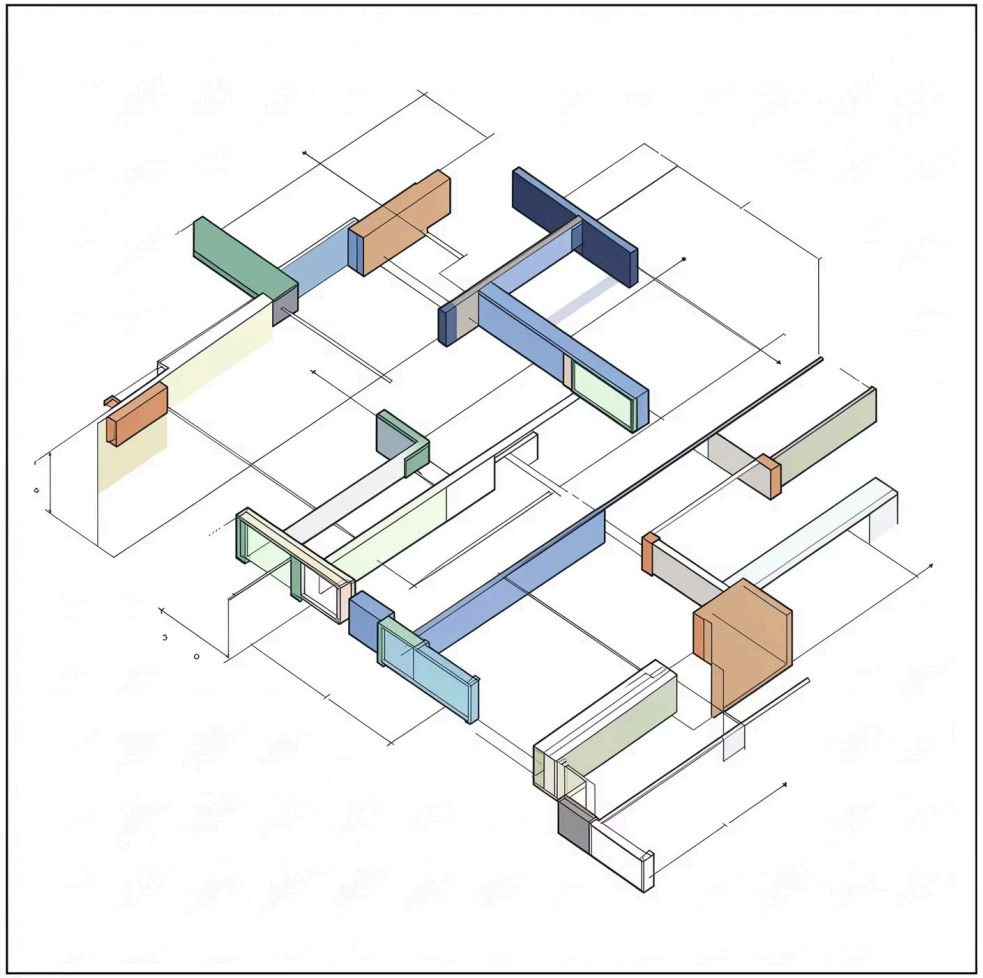
3

첫 모델 다운로드와 로딩

수 GB 모델 파일의 초기 다운로드가 가장 큰 병목. 이후 캐싱으로 완화된다

snapshot-first 전략과 offscreen preload가 이 병목들을 완화하는 핵심 설계 선택이다.

설계적으로 잘한 부분 정리



1 **브라우저 제약을 구조로 반영**

제약을 억지로 감추지 않고 3분할 컨텍스트로 자연스럽게 수용했다.

2 **관심사 명확한 분리**

모델·권한 API·DOM 작업·UI가 각자의 파일과 컨텍스트로 분리되어 있다.

3 **메시지 기반 tool RPC 추상화**

에이전트와 실행 환경의 결합도를 낮춰 교체 가능성을 높였다.

4 **UX 최적화 요소 포함**

preload, streaming, progress 표시가 모두 구현되어 있다.

기술적으로 아쉬운 부분 정리

run_javascript 가드레일 부재

모델이 생성한 임의 JavaScript를 승인 없이 실행. policy layer 없이는 사고 반경이 크다.

selector 품질 의존도

DOM 상호작용이 CSS selector 정확도에 크게 좌우된다. 복잡한 웹앱에서 빠르게 brittle해진다.

textual output 포맷 의존

structured function calling API가 아닌 모델 출력 텍스트 파싱에 의존. 모델 변경 시 깨질 위험.

단순한 multi-tab 지원

단일 세션 모델로 인해 병렬 탭 에이전트 운영이 불가능하다. 확장 시 session map 필요.

이 저장소가 보여주는 핵심 교훈 1

교훈 #1

브라우저 에이전트는 모델 성능만의 문제가 아니다

Gemma 4 모델이 아무리 뛰어나도 실행 컨텍스트 분리, 권한 모델, 메시지 버스, UX 스트리밍 설계가 뒷받침되지 않으면 실제로 작동하는 에이전트가 되지 않는다. **실제 제품성은 tool reliability와 orchestration 품질에서 갈린다.**

실행 컨텍스트 분리

브라우저 제약을 설계로 수용

권한 모델

기능과 안전성의 균형

메시지 버스

컨텍스트 간 결합도 제어

UX 스트리밍

체감 성능과 신뢰 형성

이 저장소가 보여주는 핵심 교훈 2

교훈 #2

도구를 정확히 쓰는 것이 많이 쓰는 것보다 중요하다

> 모든 질문에 도구를 먼저 호출할 필요는 없다. page snapshot 같은 정적 grounding이 매우 효율적이다.

agentic system의 성능은 도구를 많이 쓰는 데서가 아니라, **필요할 때만 정확히 쓰는 데서** 나온다. Gemma-gem의 snapshot-first 전략은 이 원칙의 실용적 구현이다. 불필요한 tool call을 줄이는 것이 곧 성능과 비용 최적화다.

이 저장소가 보여주는 핵심 교훈 3

교훈 #3

온디바이스와 안전한 에이전트는 별개 설계 문제다

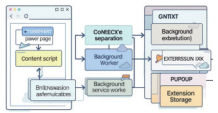
로컬 추론이 주는 것

프라이버시 보호: 데이터가 서버로 나가지 않는다.
독립성: 인터넷 없이도 동작한다.
비용 절감: API 사용료가 없다.

로컬 추론이 주지 않는 것

안전한 에이전트: 로컬이라도 브라우저 조작 권한은 여전히 강력하다.
자동화된 destructive action은 여전히 위험하다.
policy layer 없이는 사고 반경이 크다.

개발자가 이 repo에서 배울 수 있는 것



Chrome Extension 아키텍처 분해

agent runtime에 맞게 content·background·offscreen을 분리하는 실전 패턴

WebGPU 브라우저 추론 배치

ONNX Runtime + HuggingFace Transformers.js로 Gemma 4를 브라우저에 올리는 방법

메시지 기반 tool RPC

requestId/pending map 패턴으로 비동기 tool call을 구조화하는 설계

lightweight DOM automation

CSS selector 기반 read·click·type·scroll 패턴과 한계

Agent UX 설계

state/streaming/progress를 포함한 브라우저 에이전트 UX 구성 방법

결론

Gemma-gem은 작은 코드베이스 안에 브라우저 에이전트의 핵심 문제들을 압축한 레퍼런스 구현이다. 완성형 제품이라기보다, **브라우저 안에서 온디바이스 LLM 에이전트를 어떻게 성립시킬 수 있는지**를 보여주는 기술 데모이자 설계 샘플이다.

핵심 가치 1

브라우저 내 Gemma 4 WebGPU 추론

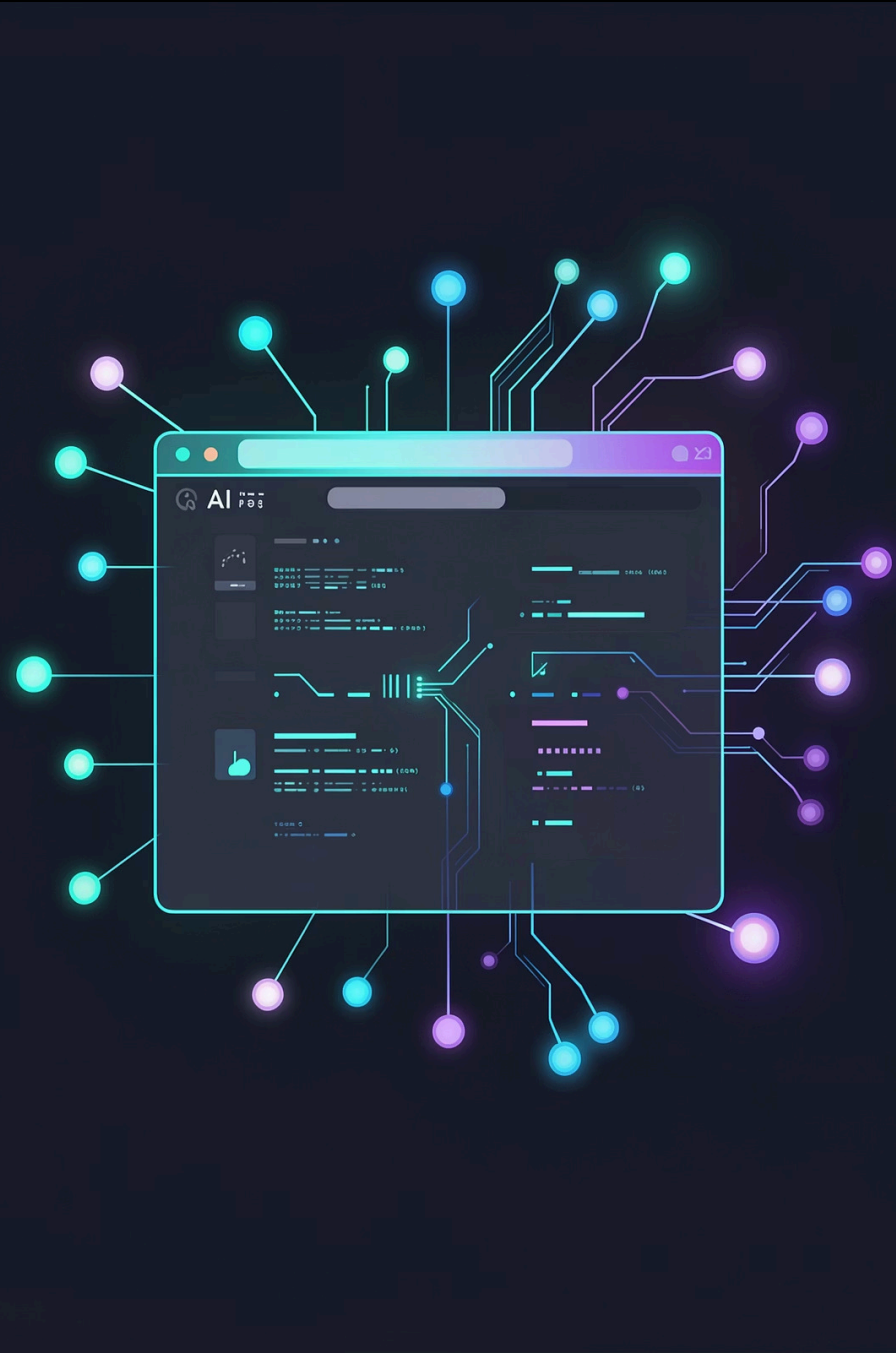
핵심 가치 2

메시지 기반 tool-using agent loop

핵심 가치 3

페이지 밀착형 로컬 UX

제품화 다음 단계: policy layer 설계, 더 견고한 DOM targeting, 안전한 action approval, multi-session orchestration, structured output robustness



유민수 개발자

AI 솔루션 개발 · AX 교육 · 기업 AI 도입 컨설팅

010-2773-2165

개발문의 교육문의 AX도입문의