Adding the PageAgent AI Assistant to Your Website

A practical, copy-paste implementation guide — with an on-demand toggle launcher · EVOKNOW.AI

PageAgent (by Alibaba, github.com/alibaba/page-agent) is a JavaScript in-page GUI agent: a small script that adds an AI assistant which can read and drive your web page through the DOM using natural language. This guide shows exactly how we wired it into a live site so it loads in English, uses the free built-in demo model, stays hidden until the visitor wants it, and opens from a single glowing button in the navigation.

What you'll end up with

- The agent loads on every page but shows **no panel by default**.
- A glowing wand icon in the top nav **toggles** the assistant open and closed.
- English UI, powered by the demo build's **free built-in model** (no API key required to try it).
- A Content-Security-Policy that permits the script and its backend — nothing else.

Step 1. Load the script — at the end of `<body>`

Add the demo build just before the closing `</body>` tag. Two rules matter:

- It must run **after** `document.body` exists — the agent appends its panel to the body. Placing it in `<head>` throws `Cannot read properties of null (reading 'appendChild')`.
- Do **not** add `defer`. The script reads its configuration from `document.currentScript.src`, which is `null` for deferred scripts — deferring silently drops your settings and falls back to the Chinese demo defaults.

```
<!-- Load at end of <body>; NOT deferred (config comes from the script URL). -->
<script
  src="https://cdn.jsdelivr.net/npm/page-agent@1.10.0/dist/iife/page-agent.demo.js?lang=en-US&showPanel=false"
  crossorigin="true"></script>
```

Configuration lives in the URL query string

- `lang=en-US` — UI language (default is `zh-CN`). The option key is `lang`, not `language`.
- `showPanel=false` — initialize the agent but keep the panel hidden (we open it on demand).

- `model`, `baseURL`, `apiKey` — optional. Omit them to use the demo build's free built-in model. Provide all three to point at your own OpenAI-compatible endpoint (Qwen/DashScope, OpenAI, DeepSeek, etc.).

Step 2. Allow it in your Content-Security-Policy

If your site sends a CSP (it should), the browser will block both the script and its network calls until you allow them. Add the jsDelivr host to `script-src` and the model backend to `connect-src`:

```
Content-Security-Policy:
  default-src 'self';
  script-src 'self' https://cdn.jsdelivr.net;
  connect-src 'self' https://page-ag-testing-ohftxirgbn.cn-shanghai.fcapp.run;
  style-src 'self' 'unsafe-inline';
  img-src 'self' data:;
```

> **Symptoms if you skip this:** "Loading the script … violates the following Content Security Policy directive: script-src 'self'" (missing `script-src`), or a retry loop of "InvokeError: Network request failed" once you send a message (missing `connect-src`). The `connect-src` host above is the free demo backend; change it to your own API host if you supply `baseURL`.

> Inline `<script>` blocks are also blocked by `script-src 'self'`. Put all custom JavaScript (Step 4) in a file served from your own origin — not inline — or the browser will refuse to run it.

Step 3. Add the glowing launcher button

Place a button in your navigation. Inline `<style>` is fine (allowed by `style-src 'unsafe-inline'`); the JavaScript is not (Step 4).

```
<button class="pa-btn" id="pa-toggle" type="button"
        aria-label="Open AI assistant" title="Ask the AI assistant">
  <i class="fa-solid fa-wand-magic-sparkles" aria-hidden="true"></i>
</button>

<style>
  .pa-btn{background:none;border:none;cursor:pointer;color:#8b5cf6;font-size:1rem;
    padding:6px;display:inline-flex;align-items:center;border-radius:999px;
    transition:transform .2s}
  .pa-btn:hover{transform:scale(1.1)}
  .pa-btn i{animation:pa-glow 2.4s ease-in-out infinite}
  @keyframes pa-glow{0%,100%{opacity:.8;filter:drop-shadow(0 0 4px currentColor)}
    50%{opacity:1;filter:drop-shadow(0 0 12px currentColor)}}
  @media (prefers-reduced-motion:reduce){.pa-btn i{animation:none;
```

```
    filter:drop-shadow(0 0 6px currentColor)}}
</style>
```

Step 4. Wire the toggle (in a script file, not inline)

The demo build exposes the live instance as `window.pageAgent`. Its panel has `show()` (sets `display:block`) and `hide()` (sets `display:none`) on a wrapper with the stable id `#page-agent-runtime_agent-panel`. Reading that element's display makes a reliable toggle that stays in sync with the panel's own close (×) button:

```
// in your site's bundled/external JS (served from your own origin)
function initPageAgent() {
  var btn = document.getElementById('pa-toggle');
  if (!btn) return;
  btn.addEventListener('click', function () {
    var pa = window.pageAgent;
    if (!pa || !pa.panel) return;
    var el = document.getElementById('page-agent-runtime_agent-panel');
    var open = !!el && el.style.display && el.style.display !== 'none';
    if (open && typeof pa.panel.hide === 'function') pa.panel.hide();
    else pa.panel.show();
  });
}
document.addEventListener('DOMContentLoaded', initPageAgent);
```

> **Why read the DOM instead of a flag?** The panel object doesn't expose a reliable public "is-open" property in this build, but the wrapper's inline `display` is authoritative and is updated by both `show()` / `hide()` and the panel's own ×. Reading it means one click opens, the next closes, and the button never drifts out of sync.

Going to production

- **Free demo model:** omitting `model` / `baseURL` / `apiKey` uses a public shared test endpoint — great for trying it, but with no uptime or rate guarantees.
- **Your own model:** supply all three params. Remember any `apiKey` on the script URL is visible in page source — for real deployments, proxy the calls through your own backend and point `baseURL` at that proxy, then allow the proxy host in `connect-src`.
- **Keep the version pinned** (`page-agent@1.10.0`) so a future release can't change behaviour under you.