

MBR Performance

The complete user guide to all-in-one WordPress speed optimisation — core controls, scripts, styles, fonts, images and database housekeeping, explained one setting at a time.

Version 1.19.0

For WordPress 5.9 – 7.0 · Requires PHP 7.4 or newer

Compatible with classic and Full Site Editing (block) themes, all major page builders, multisite networks and WooCommerce.

This guide walks through every settings tab in order. Read the **Getting Started** and **How the Settings Work** sections first, then dip into the tab you need. New to the plugin? Start with the **Doctor** — it tells you which settings will actually help your site.

Contents

| | |
|-----------|----------------------------------|
| 1 | Getting Started |
| 2 | How the Settings Work |
| 3 | The Performance Doctor |
| 4 | Core Features |
| 5 | JavaScript |
| 6 | CSS |
| 7 | Fonts |
| 8 | Preloading & Speculative Loading |
| 9 | Lazy Loading |
| 10 | Database |
| 11 | WebP / AVIF Images |
| 12 | Server |
| 13 | Diagnostics |
| 14 | Orphaned Media |
| 15 | WooCommerce |
| 16 | Multisite Networks |
| 17 | Recommended Workflow |
| 18 | Troubleshooting & FAQ |
| 19 | Glossary |

Every tab in this guide follows the same shape: a short explanation of what the tab is for, a description of each setting group, and clear guidance on what to switch on, what to leave alone, and what to test.

1 Getting Started

MBR Performance is an all-in-one optimisation plugin that gives you complete, transparent control over how WordPress builds and delivers your pages. Rather than a single "make it fast" button, it groups dozens of individual optimisations into clearly-labelled tabs so you can apply exactly what your site needs and nothing it doesn't.

Design principles

- **Complementary, not competitive.** The plugin does no page caching of its own. It sits happily alongside a caching plugin or host-level cache and concentrates on the optimisations those tools often leave on the table.
- **Safe by default.** Almost every feature ships switched off. Nothing changes on your site until you deliberately enable it, so an install or update never alters how your pages render until you choose to.
- **One setting at a time.** Features are deliberately granular. You enable one, test the front end, and move on — so if something does change unexpectedly, you always know which switch caused it.
- **Self-contained.** No external CDN dependency for the plugin's own assets, and no telemetry. The only outbound connections are optional and clearly documented (Google Fonts, YouTube and Vimeo previews — see the relevant tabs).

Installing the plugin

If you downloaded the plugin as a ZIP file:

- In WordPress admin, go to **Plugins** → **Add New** → **Upload Plugin**, choose the ZIP, then click **Install Now**.
- Alternatively, unzip it and upload the `mbr-performance` folder to `/wp-content/plugins/` via SFTP or your host's file manager.
- Click **Activate**.
- The plugin appears as **MBR Performance** in the WordPress admin toolbar at the very top of the screen. Hover it to jump straight to any tab.

TIP

Before you change anything

Take a full backup of your site (files and database). Then, if you possibly can, try new settings on a staging copy first. Enable one feature, save, and check the front end of your site in a fresh browser tab before moving to the next.

System requirements

| Requirement | Detail |
|---------------|---|
| WordPress | 5.9 or newer (tested up to 7.0). Works on classic and Full Site Editing block themes. |
| PHP | 7.4 or newer. AVIF image conversion additionally needs PHP 8.1+ with GD AVIF support, or Imagick 7.0.25+. |
| Server | Apache, LiteSpeed or Nginx. Browser-cache and compression rules are written to .htaccess on Apache/LiteSpeed; an equivalent snippet is shown for Nginx. |
| Compatibility | Elementor, Beaver Builder, Divi, Oxygen, Bricks and WPBakery. Optimisations are automatically suspended inside editor and preview modes. |

2 How the Settings Work

All of the plugin's controls live on one settings screen split into a row of tabs. The order below is the order the tabs appear on screen, and the order this guide follows.

| Tab | Purpose |
|----------------|--|
| Doctor | Analyses a real page and tells you which settings will help — the best place to start. |
| Core Features | WordPress-level switches: emojis, embeds, REST API, feeds, HTML minify and more. |
| JavaScript | Defer, delay, move-to-footer, minify, combine, jQuery handling. |
| CSS | Used CSS, async CSS, minify, combine, block styles, Google Fonts handling. |
| Fonts | Self-host Google Fonts, preload, subset, font-display, preconnect. |
| Preloading | Preload critical images, fetch priority, Early Hints, speculative loading. |
| Lazy Loading | Native image lazy-load, iframe/video lazy-load, exclusion rules. |
| Database | Revisions, drafts, trash, spam, transients, table optimisation, scheduling. |
| WebP / AVIF | Next-gen image conversion, bulk converters, image sizing & dimensions. |
| Server | Browser-cache headers and text compression via .htaccess. |
| Diagnostics | Autoloaded-options audit, WP-Cron viewer, caching-plugin conflict detector. |
| Orphaned Media | Find and safely remove unused Media Library files. |
| WooCommerce | Store-specific optimisations (only active when WooCommerce is installed). |

Saving, resetting and tooltips

- Each tab has its own **Save Changes** button. Saving applies the settings on that tab immediately.
- Most settings carry a small ? tooltip. Hover it for a one-line explanation and any compatibility warnings specific to that option.
- A **Reset** control returns settings to their safe defaults if you need a clean slate.
- Tools that act on files or the database (bulk image conversion, database cleanup, orphaned-media deletion) have their own action buttons and progress bars, and run when you click them — not on save.

NOTE

The golden rule

Change one thing, save, then load the front end of your site in a clean browser tab (ideally one where you are logged out, since some optimisations are skipped for logged-in administrators). If everything looks right, move on. If something looks off, switch that one setting back. Working this way means a problem is never more than one toggle away from a fix.

3 The Performance Doctor

The Doctor is the recommended starting point. Instead of presenting a wall of switches, it loads a real front-end page from your site, measures what is actually slowing it down, and recommends — in priority order — which settings will help. Crucially, it also tells you which settings to leave off, and it skips anything you have already enabled.

What it checks

- **Render-blocking split.** The single most decisive factor: is your first paint being held up mainly by CSS or by JavaScript? The Doctor diagnoses the balance and points you to the CSS tab or the JavaScript tab accordingly, rather than letting you guess.
- **Image issues.** It flags images missing width/height attributes (which cause layout shift), JPEG/PNG images that a next-gen format would shrink, and below-the-fold images that aren't being lazy-loaded — each routed to the exact setting that fixes it.
- **Multiple templates.** It auto-detects your key page types — home page, blog index, a single post, a page, and (if present) the WooCommerce shop and a product page — then aggregates the findings into site-wide recommendations versus page-specific ones.

Using the Doctor

- Open the **Doctor** tab and run a scan. A first-run nudge will steer you here automatically the first time you open the plugin; it dismisses itself once you've run a scan.
- Each recommendation links straight to the relevant setting, so you can act on it without hunting through tabs.
- When you're done, you can generate a branded, print-ready **PDF report** of the scan to hand to a client. The report is produced in your browser, so it adds no weight to the plugin and works on any host.

TIP

Advisory only

The Doctor never changes any settings by itself. It only recommends. You stay in full control of what actually gets switched on — run a scan, read the advice, then decide. Re-run it after making changes to confirm the improvement.

NOTE

Where the real win is

If the Doctor reports that your render-blocking is mostly **CSS**, the fix is on the CSS tab (Used CSS / async). If it's mostly **JavaScript**, the fix is on the JavaScript tab (Defer / Delay). Spending effort on the wrong one is the most common way to get a disappointing result, which is exactly what the Doctor is there to prevent.

4 Core Features

The Core tab switches off WordPress features and front-end clutter that most sites never use. None of these are dramatic speed wins on their own, but together they trim requests, remove inline scripts and reduce your site's surface area. The settings are grouped into sections.

Scripts & Styles

| Setting | What it does | When to use |
|--------------------------|--|---|
| Disable Emojis | Removes the emoji-detection script and inline styles | Safe on almost any site that doesn't rely on legacy emoji rendering |
| Disable Dashicons | Stops the Dashicons icon font loading on the front end | Only when no front-end feature needs it; may affect the admin bar |
| Disable Embeds | Removes oEmbed, so YouTube/Twitter URLs no longer auto-embed | When you paste media as blocks rather than bare URLs |
| Disable jQuery Migrate | Drops the compatibility shim for old jQuery code | Only if no plugin/theme needs it — test thoroughly |
| Remove Global Styles | Strips the inline global-styles CSS block | Classic themes only — auto-skipped on block themes |
| Conditional Block Styles | Loads block CSS only when a block is actually present | Generally safe and worthwhile |

WARNING

Remove Global Styles and block themes

On a Full Site Editing (block) theme — Twenty Twenty-Two onwards and similar — the inline global-styles block is what renders your colours, fonts, spacing and layout on the public front end. The plugin automatically skips this optimisation when a block theme is active, but the safest approach is to leave it off unless you run a classic theme.

WordPress Features

- **Disable XML-RPC** — closes a legacy remote-access endpoint. Improves security but can break the Jetpack connection and some mobile apps.
- **Remove WordPress version** — hides the version number from the page head and feeds (a small security-hardening measure).
- **Disable RSD / shortlinks / self-pingbacks** — removes rarely-needed discovery and ping features from the head.
- **Disable RSS feeds**, or just hide feed links — turn feeds off entirely, or keep them working but remove their discovery links from the head.
- **REST API control** — choose between leaving the REST API fully open, disabling it for non-admins, or disabling it when logged out. An allowlist field lets you keep specific namespaces public (for example a contact form, store API or chat widget) even when the API is otherwise restricted.
- **Disable AI Features (WordPress 7.0+)** — WordPress 7.0 ships a built-in AI Client, Abilities API and a Connectors screen. They stay dormant until you connect a provider, but this toggle switches the whole subsystem off using core's own kill switch. It has no effect on WordPress 6.x, so it's safe to leave on across mixed-version sites.

TIP**REST API: don't go all-or-nothing**

If you need to lock the REST API down, use the allowlist rather than a blanket block. Common public namespaces to keep open include contact-form and store endpoints. Blocking everything is the usual cause of "the contact form / store suddenly stopped working" after hardening.

Advanced Performance — Minify HTML

Also on the Core tab, **Minify HTML** strips comments and collapses excess whitespace from your page output. It is carefully built to preserve the contents of `script`, `style`, `pre`, `textarea` and inline SVG blocks, to honour IE conditional comments, and to skip pages that embed a complete nested HTML document (such as a full landing page inside a page-builder HTML widget). AMP, REST, AJAX, feed, embed and customiser-preview responses are never touched, and if anything fails to round-trip the original output is restored wholesale. Off by default.

NOTE**Modest by design**

Once gzip or brotli compression is active (see the Server tab), HTML minification saves relatively little. Treat it as a small finishing touch rather than a headline optimisation, and leave it off if you ever see odd whitespace-sensitive layout.

5 JavaScript

JavaScript is the most common cause of a slow or janky first load, so this tab is where many of the biggest wins live — but it's also where careless changes break sites. Enable one option at a time and test interactive elements (sliders, menus, forms, cart) after each.

Loading strategy

| Setting | What it does | When to use |
|------------------------|---|---|
| Defer JavaScript | Lets the page render first, then runs scripts in order | A strong, generally-safe first step; each script honours its own exclusion list |
| Move scripts to footer | Relocates scripts below the page content | When a theme/plugin enqueues scripts in the head unnecessarily |
| Delay JavaScript | Holds non-critical scripts until first user interaction | Excellent for analytics and third-party widgets; set a timeout fallback |
| Async JavaScript | Runs scripts as soon as they load, out of order | For independent scripts that don't depend on others |

TIP

Delay is your friend for third-party scripts

Chat widgets, analytics and tag managers rarely need to run before the visitor interacts. Delaying them until the first scroll, tap or move can transform a PageSpeed score with no visible downside. Keep a timeout so they still fire on pages where the visitor never interacts.

jQuery handling

- **Defer jQuery** — defers the core jQuery library along with everything else. Test first, as some inline scripts expect jQuery to be ready immediately.
- **Remove jQuery** — for modern sites that don't depend on it. A built-in **test mode** lets you check whether anything breaks before committing.

Minify & combine

- **Minify inline JS** — shrinks inline script blocks.
- **Combine JavaScript** — merges multiple local scripts into one cached bundle to cut HTTP requests, processed separately for the head and footer groups. For safety it only combines "pure" scripts: any script carrying inline or localised data (which routinely includes per-request security nonces), an async/defer strategy, or a conditional, is left alone. Scripts in your Defer, Delay and Exclude lists are also respected.
- **Remove script versions** — drops the `?ver=` query string for better edge caching.

NOTE**Why Combine JS merges fewer files than Combine CSS**

This is expected, not a fault. CSS can be merged freely, but a script carrying inline data (often a security nonce) can't be safely baked into a shared cached file, so it breaks the run and stays separate. You still get the big win — jQuery and the cluster of vanilla libraries folded together — just with more standalone files than on the CSS side. If a combined script misbehaves (typically one that loads its own workers via its script URL), add it to the Exclude list.

Each tab shows how many combined files are currently cached and offers a one-click **Clear combined cache** button. Bundles rebuild automatically when your settings or assets change; the button is just a manual flush.

6 CSS

If the Doctor reports that your render-blocking is mostly CSS, this is the tab that fixes it. The headline feature is Used CSS (Mode A).

Used CSS (Mode A)

With **Generate Used CSS** enabled, the plugin works out which CSS rules each page actually uses, inlines just those rules in the page head for an instant first paint, and loads your full stylesheets asynchronously behind them. This removes render-blocking unused CSS from the critical path.

It is the **safe** variant of unused-CSS removal: the original stylesheets are never deleted, only deferred. So if the analysis misses a rule — for example one used by a widget that JavaScript injects after load — the full stylesheet still arrives a moment later and corrects it. Used CSS is generated in the background after the first visit to each page and cached per URL, so the first visitor triggers generation and later visitors get the optimised version. If your host runs a full-page cache, the relevant URL is purged automatically after generation so the optimised version is what gets cached.

When Used CSS helps (and when it doesn't)

- It earns its place on **CSS-heavy** sites — a large theme plus a page builder plus several plugins each loading their own stylesheet.
- It only helps if your render-blocking is genuinely caused by CSS. Check PageSpeed Insights: if the render-blocking list is mostly stylesheets, Used CSS helps; if it's mostly JavaScript, the win is on the JavaScript tab instead.
- On a site already light on CSS, the gain may be small.

TIP

Getting the best from Used CSS

Leave it off by default and test on a staging copy first. If a particular stylesheet must always load in full, add part of its URL to the **Exclude from optimization** field. There's no need to also enable Async CSS — Used CSS handles delivery itself and automatically suppresses the standalone async layer to avoid the two conflicting.

Other CSS settings

| Setting | What it does | When to use |
|----------------------|--|---|
| Async CSS | Loads stylesheets without blocking render (preload + onload) | When you're not using Used CSS; Used CSS supersedes it |
| Minify CSS | Strips whitespace and comments from CSS | Generally safe; also minifies the combined bundle |
| Combine CSS | Merges adjacent same-origin stylesheets into one bundle | Cuts requests; stood down automatically when Used CSS is active |
| Preload Combined CSS | Emits an early preload hint for each bundle | Only with Combine CSS on; auto-skipped when Async CSS is on |
| Remove CSS versions | Drops the ?ver= query string from stylesheet URLs | Better edge caching |

| Setting | What it does | When to use |
|-------------|---|--|
| CSS scanner | Reports unused styles for manual review | Investigation aid, not an automatic change |

NOTE**Used CSS or Combine CSS — not both**

These two are alternatives. Used CSS (Mode A) already owns CSS delivery, so when it's on, Combine CSS is automatically stood down. The Doctor will recommend one or the other, never both.

7 Fonts

Web fonts are a frequent and easily-fixed source of slow first paint and privacy headaches. This tab lets you self-host Google Fonts, control how they load, and remove them entirely where they aren't needed.

Self-hosting Google Fonts

Enable **Self-host Google Fonts** and download a font from this tab. Your server fetches the stylesheet and font files from Google once, stores them locally, and serves them from your own domain thereafter — so your visitors' browsers never contact Google. This removes a third-party connection, speeds up font delivery, and helps with GDPR concerns around Google Fonts.

NOTE

What gets sent to Google, and when

The download happens only at the moment you trigger it in the admin area. Your server sends the requested font family name(s) and standard request information (its IP address and user agent) to Google's font servers. No website-visitor data is sent. Once downloaded, fonts are served entirely from your own server.

Loading and display controls

- **Preload critical fonts** — tells the browser to fetch your most important fonts early, reducing the flash of invisible/unstyled text.
- **Preconnect to font domains** — warms up the connection to a font host before it's needed (useful if you keep some fonts remote).
- **Font subsetting** — strips unused characters to shrink font files.
- **Font-display strategy** — choose *swap*, *block*, *fallback* or *optional* to control how text behaves while a font loads (*swap* is the usual choice for showing text immediately).
- **Font Awesome optimisation** — trims icon-font overhead where it applies.
- **Manual font management** — add and manage your own font files directly.

Removing Google Fonts entirely

Disable Google Fonts strips Google Fonts from your pages completely — both the `googleapis.com` and `gstatic.com` requests. As of this version it also catches Google Fonts that a theme hardcodes directly into the page (header link tags, preconnects, inline `@font-face` and `@import`) which bypass WordPress's normal enqueue system, via a guarded final-output pass. There's also a dedicated **Elementor Google Fonts** control and a **Clear font cache** button.

TIP

A practical font workflow

For most sites: enable Self-host Google Fonts, download the families your theme actually uses, set font-display to swap, and preload only the one or two fonts visible above the fold. Then disable any remaining remote Google Fonts so nothing slips out to Google in the background.

8 Preloading & Speculative Loading

Where lazy loading defers what isn't needed yet, preloading does the opposite: it fetches the few resources the browser will need immediately, earlier than it otherwise would. Used sparingly, it improves perceived speed and your Largest Contentful Paint (LCP).

Preloading critical resources

- **Preload critical images** — fetch your LCP/hero image early so the most important visual appears sooner. Use this only for the genuine above-the-fold image; preloading everything defeats the purpose.
- **Fetch Priority** — automatically assigns high priority to the first image, lets you mark additional critical images via CSS selectors, and can disable WordPress core's own fetch-priority behaviour if it's conflicting.
- **Cloudflare Early Hints (HTTP 103)** — supports the Early Hints mechanism so a supporting host/CDN can start delivering resource hints before the full response is ready.

Speculative loading

Speculative loading makes the *next* navigation feel instant by fetching or even rendering the likely next page in advance.

| Setting | What it does | When to use |
|-----------------|---|---|
| Prefetch mode | Fetches the next page's resources ahead of the click | Low-risk, broad speed-up for navigation |
| Prerender mode | Fully renders the next page in the background | Strongest effect; use moderate eagerness to avoid waste |
| Eagerness level | Conservative, moderate or eager — how aggressively to speculate | Start conservative/moderate and increase if needed |
| Auto mode | Lets the browser decide the optimal strategy | A sensible hands-off default |

NOTE

Don't over-preload

Preloading and prerendering consume bandwidth and CPU. Reserve image preloading for your single hero image, and keep speculative eagerness moderate unless you've confirmed a clear benefit. Aggressive prerendering on a site with many outbound links can waste resources fetching pages nobody visits.

9 Lazy Loading

Lazy loading defers off-screen images, iframes and embedded videos until the visitor scrolls near them, so the initial page weighs far less and loads faster. The key to using it well is excluding the images that must appear immediately.

What you can lazy-load

- **Native image lazy loading** — uses the browser's built-in lazy-load for images.
- **iframes and embedded videos** — defers YouTube, Vimeo and other embeds until needed.
- **Video facade** — replaces a heavy embedded player with a lightweight thumbnail and play button; the real player loads only when the visitor clicks. This saves a large amount of JavaScript on first load and stops the embed setting cookies until interaction.

NOTE

How the video facade fetches previews

To show the placeholder thumbnail, the visitor's browser requests the preview image from YouTube (i.ytimg.com) or Vimeo's public API. That request includes the visitor's IP address and the video ID. The full player and any video-provider cookies load only if the visitor clicks play. The facade is keyboard-accessible (Enter/Space).

Excluding critical images

The most important setting here isn't what you lazy-load, but what you **exclude**. Your logo and above-the-fold hero image should load immediately — lazy-loading them hurts your LCP. The plugin lets you exclude images by:

- CSS selectors
- Class names and IDs
- Data attributes
- Keywords found in the image's src or class
- Parent element selectors

TIP

The rule for lazy loading

Lazy-load everything below the fold; never lazy-load what's above it. After enabling, scroll your home page and a couple of key templates: anything visible without scrolling — logo, hero, first product image — should appear instantly. If one fades in late, add it to an exclusion rule.

10 Database

Over time a WordPress database accumulates clutter — old revisions, spam, abandoned drafts, expired transients — that bloats tables and slows queries. This tab cleans it up, on demand or on a schedule.

What it cleans

- **Post revisions** — trim to a configurable number kept per post (uses the proper WordPress filter, so the limit actually takes effect).
- **Auto-drafts and trash** — delete old auto-saved drafts and emptied-trash items past a configurable age.
- **Spam and unapproved comments** — remove spam and pending comments older than a set age.
- **Orphaned metadata** — clear post, comment, term and relationship metadata left behind by deleted content.
- **Transients** — purge expired transients (handles network/site transients on multisite).

Table maintenance

- **Optimise tables** — reclaim space and defragment database tables.
- **Convert MyISAM to InnoDB** — move legacy tables to the modern storage engine for better concurrency and reliability.
- **Repair tables** — attempt repair on tables flagged as damaged.

Scheduling

Set the cleanup schedule to **daily**, **weekly** or **manual**. The scheduled job re-aligns itself automatically to whatever you choose, runs all the enabled cleanups together, and writes a **Last Auto-Cleanup** log to this tab so you can see exactly what was removed and when. A **Run Auto-Cleanup Now** button lets you trigger it on demand.

WARNING

Cleanup is permanent — back up first

Deleting revisions, drafts, trash and comments cannot be undone from inside the plugin. Take a database backup before your first cleanup, and start with conservative retention values (for example keep more revisions and a longer trash age) until you're comfortable with what's being removed.

11 WebP / AVIF Images

Images are usually the heaviest thing on a page. This tab converts them to modern, far smaller formats and fixes the most common image-related PageSpeed warnings. It's one of the highest-impact tabs in the plugin.

Next-gen formats: WebP and AVIF

The plugin converts your JPG, JPEG and PNG images to **WebP** and, where your server supports it, **AVIF** — typically 20–30% smaller than WebP again at equivalent perceived quality. Converted images are delivered through an HTML `<picture>` wrapper that offers AVIF first, then WebP, then the original JPEG/PNG as a fallback, so each browser automatically picks the first format it supports. Your originals are never thrown away.

- **Convert on upload** — every new image is converted automatically as it's added.
- **Bulk converters** — convert your existing Media Library in batches, with a progress bar and per-image AJAX so large libraries don't time out. Separate WebP and AVIF runs.
- **Compression level** — set quality from 1–100 to balance size against fidelity.
- **Smart skip** — if a converted file would be larger than the original, it's skipped automatically.
- **Conversion history** — a table showing each image with its WebP and AVIF sizes and the saving achieved, plus a registry-driven **Revert All** that deletes only the files the plugin created, never your originals.
- **Server diagnostics** — confirms GD/Imagick support so the AVIF tools only appear when your host can actually encode AVIF.

NOTE

Apache / LiteSpeed rewrite option

On Apache or LiteSpeed you can optionally serve WebP transparently via .htaccess rewrite rules. On other servers the `<picture>` delivery handles format selection in the browser instead — either way, unsupported browsers always receive the original.

Image Sizing & Dimensions

This section tackles two more PageSpeed warnings: "Properly size images" and "Ensure images have explicit width and height".

- **Resize on upload** — automatically scales oversized uploads down to a configurable maximum (default 2560px), preserving aspect ratio via the WordPress core pipeline.
- **Inject width & height** — adds missing dimension attributes to front-end images, which lets the browser reserve space and reduces Cumulative Layout Shift (CLS). Works on post content, Gutenberg blocks, Elementor widgets, attachment images and thumbnails. Dimension lookups are cached per URL for a week; external images, SVGs and data URIs are skipped.
- **Bulk resize** — scan for existing images above your maximum, then downscale them in place with a progress bar and live log. Sub-sizes are regenerated and stale WebP files cleaned up automatically afterwards.
- **decoding="async"** and **EXIF stripping** — let images decode off the main thread, and remove camera/GPS metadata from new JPEGs (ICC colour profiles are preserved).

WARNING**Bulk resize overwrites files permanently**

Unlike resize-on-upload, the bulk resize tool rewrites existing files on disk and cannot be undone automatically. Take a full backup before running it, and let the scan show you what will change before you start the resize.

12 Server

Two of the most common PageSpeed Insights warnings — "Serve static assets with an efficient cache policy" and "Enable text compression" — are fixed at the web-server level. This tab writes the necessary rules for you.

Browser-cache headers

Adds Expires and Cache-Control headers so returning visitors' browsers reuse your static assets instead of re-downloading them: roughly one year for images and fonts, thirty days for CSS and JavaScript. On Apache and LiteSpeed the rules are written to `.htaccess` inside a clearly-marked `MBR Browser Cache` block.

Text compression

Enables Brotli and Gzip compression for text resources (HTML, CSS, JavaScript, SVG and so on), typically cutting their transfer size by a large margin. Written to `.htaccess` in an `MBR Compression` block.

Nginx hosts

Nginx doesn't use `.htaccess`. The plugin detects your web server and, on Nginx, shows you an equivalent configuration snippet to paste into your server block instead. Both marker blocks are removed cleanly if you deactivate the plugin.

NOTE

Check with your host's cache

Many managed hosts already set cache headers and compression at the server or CDN level. If yours does, you may not need these toggles — and the Diagnostics tab's conflict detector will help you spot overlap with optimisation plugins you already run.

13 Diagnostics

The Diagnostics tab is a set of investigative tools that surface performance problems WordPress otherwise hides from you.

- **Autoloaded Options Audit** — autoloaded options are loaded on every single page request, so a bloated `wp_options` table is one of the most common silent performance killers. This tool shows your total autoloaded size and the 30 largest options, with a one-click **Disable autoload** button. Protected core options (`siteurl`, `home`, `active_plugins`, `template`, `stylesheet` and the like) cannot be touched, and transients are flagged.
- **WP-Cron Viewer** — lists every scheduled event with its next run, recurrence, and a column showing whether a PHP callback is actually registered for it. Events with no callback — left behind by deactivated plugins — are flagged as **orphan** and can be unscheduled in one click. It also explains how to replace WP-Cron with a real system cron for performance-sensitive sites.
- **Caching Plugin Conflict Detector** — detects popular caching/optimisation plugins (WP Rocket, W3 Total Cache, LiteSpeed Cache, FlyingPress, WP Super Cache, Perfmatters, Autoptimize and SiteGround Optimizer) and lists exactly which MBR options overlap with each, so you never run the same combine, minify or defer pass twice.

TIP**Run the conflict detector early**

If you use any caching or optimisation plugin alongside this one, check the conflict detector before enabling Combine, Minify or Defer here. Running the same pass in two plugins is the classic cause of broken layouts and scripts — pick one tool to own each job.

14 Orphaned Media

Over the life of a site, the Media Library fills with files that nothing references any more — replaced images, abandoned uploads, leftovers from deleted posts. This tab finds them and removes them through a deliberately cautious, reversible workflow.

How detection works

A scan checks each attachment against several sources of references: its post parent, use as a featured image, appearances in post content (matched by attachment ID, shortcode reference and filename stem, so sized variants and URL-only references are caught), and a string-search across all postmeta values. You can scan images only, or extend the scan to videos, audio, documents and archives via the tab's checkboxes.

The two-tier safety classifier

- **High-confidence orphans** (zero references found anywhere) are eligible for bulk deletion.
- **Review candidates** (matched only in postmeta) must be inspected and deleted individually — this tier deliberately covers references stored in page-builder data that the scan can't fully parse.

The deletion and restore workflow

- Before anything is deleted, a staging table records the full attachment record, its postmeta and a file manifest.
- You choose a **restore window** — 7, 14, 30 or 60 days, or "keep forever". A daily job purges staging records once their window expires.
- At delete time, orphan status is **re-verified**: if an attachment has become referenced since the scan, the deletion is blocked.
- For images, the original, all sub-size variants, the scaled full-size variant and matching .webp siblings are removed together; for other media, the single file is removed.
- A per-attachment **exclusions list** permanently keeps chosen IDs off the orphan list.
- Scans run with a live progress bar, batched in chunks of 50 to avoid timeouts on large libraries.

WARNING

Restore brings back the record, not the file

Restoring within the window reinstates the database record, but the image bytes are physically deleted at the time of staging — a restored attachment must be re-uploaded if you need the file itself. Always review the candidate list carefully (especially the review tier) before bulk-deleting, and back up first.

15 WooCommerce

This tab activates only when WooCommerce is installed, and consolidates the store-specific optimisations that make the biggest difference to a shop's speed. When WooCommerce isn't present, the tab shows an inactive state so the capability stays discoverable.

Key optimisations

- **Cart fragments control** — WooCommerce fires an admin-ajax request on every page load to refresh cart fragments, which is a major drag on cached sites. Disable it site-wide, or only on non-shop pages. This is often the single biggest WooCommerce TTFB win.
- **Conditional asset loading** — dequeues WooCommerce scripts, styles, block assets, selectWoo and blockUI on pages that aren't part of the store, so your blog and landing pages don't carry shop weight.
- **Disable the password strength meter** (zxcvbn) on the front end.
- **Disable marketplace suggestions and dashboard widgets**, and prevent the heavy wc-admin React bundles from loading on non-WooCommerce admin screens.
- **Action Scheduler retention** — cap how long completed scheduled actions are kept (default 30 days, with shorter options) to stop the `actionscheduler_actions` table ballooning on busy stores.
- **One-click cleanups** — clear expired WooCommerce sessions and product/order/expired transients, with an optional weekly automated cleanup and a last-run log.

NOTE

Geolocation vs. page caching

The tab warns you if WooCommerce's default customer location is set to "Geolocate", which breaks full-page caching, or "Geolocate with page cache support", which appends a query string some caches mishandle. A direct link takes you to the WooCommerce setting to resolve it.

TIP

Test the mini-cart after enabling cart fragments

If your theme relies on a live-updating mini-cart that changes the moment an item is added, disabling cart fragments site-wide can stop it updating without a page reload. If that matters to your store, restrict the setting to non-shop pages instead, and test the add-to-cart experience.

16 Multisite Networks

On a WordPress Multisite network the plugin can be managed centrally from the Network Admin, so you set your performance policy once and apply it everywhere.

- **Network activation** — activate (and deactivate) the plugin across the whole network at once.
- **Network defaults** — manage a single set of default settings from **Network Admin** → **Settings** → **MBR Performance**.
- **Push to all sites** — apply your network defaults to every site, or a selection of sites, in one click.
- **Import from a site** — take the settings from any existing site and adopt them as the network defaults.
- **Per-site override control** — super admins decide whether individual site admins may customise their own settings or are locked to the network defaults. When overrides are locked, the per-site Save and Reset controls are disabled and a notice explains why.
- **Automatic setup for new sites** — newly-created sites in the network inherit the network defaults automatically.

TIP

A sensible network policy

Configure one representative site exactly how you want it, import those settings as your network defaults, then push to all sites. Decide deliberately whether to lock overrides: locking keeps the network consistent, unlocking lets site owners tune their own pages.

17 Recommended Workflow

Putting it all together, here is a safe order of operations that gets most sites most of the way with the least risk.

- 1 Back up.** Take a full backup, and work on a staging copy if you can.
- 2 Run the Doctor.** Let it tell you whether your render-blocking is CSS or JavaScript, and which image issues to fix. Act on its priority list rather than guessing.
- 3 Fix images first.** On the WebP / AVIF tab, bulk-convert to next-gen formats and inject missing width/height. Images are usually the biggest single win and the lowest risk.
- 4 Address render-blocking.** If the Doctor flagged CSS, enable Used CSS. If it flagged JavaScript, enable Defer and Delay non-critical scripts. Do one, test, then the other.
- 5 Tidy fonts.** Self-host Google Fonts, set font-display to swap, and preload only above-the-fold fonts.
- 6 Set lazy loading.** Lazy-load below-the-fold images and videos; exclude your logo and hero image.
- 7 Server headers.** Enable browser-cache headers and compression — unless your host or CDN already provides them (check Diagnostics).
- 8 Database housekeeping.** Trim revisions, clear clutter, and set a sensible cleanup schedule.

-
- 9** **Check for conflicts.** Run the Diagnostics conflict detector if you use any other caching/optimisation plugin, and switch off any duplicated pass.
 - 10** **Re-measure.** Re-run the Doctor and PageSpeed Insights, and click through your key templates logged out to confirm everything renders correctly.

18 Troubleshooting & FAQ

Something on my site broke after I changed a setting. What now?

Because the plugin is granular and you changed one thing at a time, the cause is the last setting you enabled — switch it back off and the problem should clear. If you're unsure which it was, reset the relevant tab to defaults and re-enable features one at a time, testing the front end after each.

Can I use this alongside a caching plugin?

Yes. The plugin does no page caching of its own and is designed to complement caching plugins and host-level caches. Where a feature overlaps with something your cache plugin also does — combine, minify, defer and so on — run it in one place only. The Diagnostics conflict detector flags the exact overlapping toggles for you.

Combine JavaScript merged far fewer files than Combine CSS. Is it broken?

No — that's correct behaviour. CSS can be merged freely, but a script carrying inline data (which often includes per-request security nonces) can't be safely baked into a shared cached file, so it's left on its own. You still get the main benefit of folding jQuery and the vanilla libraries together; you just keep more standalone files on the JS side.

Used CSS didn't speed up my site.

Used CSS only helps when your render-blocking is genuinely caused by CSS. Check PageSpeed Insights: if the render-blocking list is mostly JavaScript, the win is on the JavaScript tab (Defer/Delay) instead. On sites already light on CSS the gain is naturally small.

Does it work with my page builder?

Yes — Elementor, Beaver Builder, Divi, Oxygen, Bricks and WPBakery are all supported, and optimisations are automatically suspended inside editor and preview modes so the builder keeps working normally.

My logo or hero image fades in late.

That image is being lazy-loaded when it shouldn't be. On the Lazy Loading tab, add it to an exclusion rule (by class, ID, selector or a keyword in its filename) so it loads immediately. Never lazy-load anything visible above the fold.

Does this disable WordPress 7.0's built-in AI?

It can. WordPress 7.0 adds a core AI Client, Abilities API and Connectors screen, which stay dormant until you connect a provider. The Core tab's "Disable AI Features" toggle switches the whole subsystem off using core's own kill switch. It has no effect on WordPress 6.x.

I restored an orphaned image but the file is still missing.

Restoring reinstates only the database record. The file bytes are physically deleted when an attachment is staged for deletion, so a restored item must be re-uploaded if you need the actual file. This is why reviewing the candidate list before deleting matters.

Where are the settings?

Click **MBR Performance** in the WordPress admin toolbar at the top of the screen. Hover it to jump straight to any individual tab.

19 Glossary

| | |
|---------------------------------------|---|
| AVIF | A modern image format, typically 20–30% smaller than WebP at equivalent perceived quality. |
| CLS (Cumulative Layout Shift) | A measure of how much page content unexpectedly jumps around while loading. Adding image width/height attributes reduces it. |
| Defer / Async | Two ways to stop a script blocking page render. Defer runs scripts in order after the page parses; async runs each as soon as it loads. |
| Delay | Holding non-critical scripts until the visitor first interacts with the page, so they don't slow the initial load. |
| Fetch Priority | A browser hint marking certain resources (such as the hero image) as high priority so they load sooner. |
| LCP (Largest Contentful Paint) | How long the largest visible element takes to appear — a key perceived-speed metric. Preloading the hero image improves it. |
| Lazy loading | Deferring off-screen images and embeds until the visitor scrolls near them, reducing initial page weight. |
| Render-blocking | Resources (usually CSS or JavaScript) the browser must process before it can paint the page. Reducing them speeds first paint. |
| Speculative loading | Fetching or pre-rendering the likely next page in advance so the next click feels instant. |
| Transient | A temporary cached value stored in the WordPress database with an expiry. Expired ones accumulate and can be cleaned up. |
| Used CSS (Mode A) | Inlining only the CSS a page actually uses and loading the full stylesheets asynchronously behind it, removing render-blocking unused CSS safely. |
| WebP | A widely-supported next-gen image format, substantially smaller than JPEG/PNG at similar quality. |

Work through the tabs at your own pace, change one thing at a time, and let the Doctor guide your priorities. Used that way, MBR Performance gives you precise, transparent control over your site's speed without the guesswork.