ChatGPT

Comparison of Tailwind CSS 4.1.11 vs Tailwind CSS 3.x

Introduction: Tailwind CSS v4.1 (released 2025) is a major update from the 3.x series, bringing fundamental changes to configuration, utility class generation, build workflow, and available features. This document highlights key differences between Tailwind 4.1.11 and Tailwind 3.x, focusing on changes that affect how developers (and AI systems generating Tailwind code) use the framework. It covers changes in utility classes, configuration and theming, build/compilation process (JIT mode and more), new syntax paradigms, deprecated features, and developer experience improvements. A summary table is provided below for a quick overview, followed by detailed sections.

Summary of Major Differences

Aspect	**Tailwind CSS 3.x** (v3)	**Tailwind CSS 4.1** (v4.1.11)
Configuration	Uses `tailwind.config.js` (JavaScript) for theme, plugins, etc. <br>– Configuration is separate from CSS, required for customization [1].<br>– **Content paths** must be manually specified in config to purge unused styles [2].	**CSS-First configuration** with `@theme` directives and CSS variables. <br>– You can define theme values directly in CSS using custom properties instead of JS config [3] [4]. <br>– `tailwind.config.js` **is optional**; not auto-loaded unless explicitly imported via `@config` [5]. <br>– **Automatic content detection** scans files by default (ignoring e.g. .gitignore files) so no manual content array needed [2].
Utility Class Generation	**Fixed scales & config needed:** Only predefined values (or those added via config) generate utilities. <br>– Custom values require config or **arbitrary value** syntax (e.g. `w-[123px]`) [6]. <br>– Limited numeric ranges (e.g. grid columns up to 12 by default, z-index up to certain values). <br>– No support for arbitrary *properties* (only arbitrary values for supported properties) [7].	**Dynamic utility values:** Many classes accept **any numeric or length value** without config. <br>– **Any size numbers** for spacing, sizing, z-index, grid columns, etc., are allowed (e.g. `w-103`, `grid-cols-15`, `z-40`) and computed from a base scale [8] [9]. <br>– **Arbitrary properties** supported via `style-[property:value]` classes, allowing custom CSS properties in HTML (e.g. `style-[scroll-snap-align:start]`) [7]. <br>– Built-in support for new utility types like 3D transforms, container queries, etc., without extra plugins [10] [11].

Aspect	**Tailwind CSS 3.x** (v3)	**Tailwind CSS 4.1** (v4.1.11)
Build & JIT Workflow	**JIT mode** introduced (default in v3) to generate only used classes. <br>– **Content purge required**: developers must maintain a `content` array in config to guide JIT in removing unused CSS [2] . <br>– Uses PostCSS plugin (`tailwindcss`) plus external tools (e.g. `postcss-import`, `autoprefixer`) for CSS imports and prefixing [12] . <br>– Tailwind CLI included in core package (v3) for builds.	**Always-on JIT** with improved performance (new engine). <br>– **Tree-shaking by default:** Unused styles are automatically removed based on scanned content (no extra setup needed) [13] . Content paths are auto-detected if not configured [2] . <br>– **Faster builds:** v4's engine yields ~3–5× faster full builds and up to 100× faster incremental rebuilds [14] [15] . <br>– **Simplified build setup:** A dedicated PostCSS plugin (`@tailwindcss/postcss`) handles imports/prefixes internally (no separate autoprefixer needed) [16] [12] . First-party **Vite plugin** (`@tailwindcss/vite`) is available for direct integration [17] . The CLI is now in `@tailwindcss/cli` package [18] .

Aspect	**Tailwind CSS 3.x** (v3)	**Tailwind CSS 4.1** (v4.1.11)
Syntax & Usage	**Directives:** Uses `@tailwind` directives in CSS to include base/components/utilities (e.g. `@tailwind base;`) in the input CSS file [19] . <br>– **Variants chaining:** Multiple variants on a class are processed right-to-left (the last variant in the string applies outermost) [20] . <br>– **No native nesting:** CSS nesting requires a PostCSS plugin (if used) – not part of Tailwind itself. <br>– **Container queries:** Not supported in core (needed an official plugin in v3.3) [21] . <br>– **Arbitrary variants:** Some support via special syntax (e.g. `focus-visible:` was built-in; arbitrary attribute selectors limited without config).	**Directives:** Uses standard CSS import – `@import "tailwindcss";` – instead of `@tailwind` directives [19] . Tailwind 4 also introduces new at-rules: `@theme`, `@utility`, `@variant`. <br>– **CSS-first customization:** Use `@theme` in CSS to set design tokens (CSS variables) for colors, spacing, etc., instead of JS theme in config [3] [22] . Use `@utility` to define custom utilities in CSS (replacing the old `@layer utilities` mechanism) [23] . Define custom variants with `@variant` / `@custom-variant` rules (no JS plugin needed for basic cases) [22] [24] . <br>– **Variant grouping order:** Stacked variants now apply **left-to-right** (more intuitive, matching CSS reading order) [20] . Tailwind 4.1 also supports **arbitrary variant groups** for cleaner classnames – e.g. `[&:hover,&:focus]:bg-blue-500` applies the same utility on hover or focus together [25] . <br>– **Native CSS nesting:** Fully supported (via Lightning CSS) so you can nest selectors in your CSS when using `@apply` or custom CSS, following standard `&` syntax [26] . <br>– **Container queries:** Now a first-class feature using the CSS `@container` rule. Mark elements as containers by adding `@container` in the class list, and use new variants like `@sm:` or `@max-md:` to apply utilities based on container width [27] [28] (no separate plugin needed).

Aspect	**Tailwind CSS 3.x** (v3)	**Tailwind CSS 4.1** (v4.1.11)
Utility Classes & Features	**Available utilities (v3):** Comprehensive 2D layout and design utilities, but *no core text-shadow, mask, 3D, etc.*. <br>– **Color palette:** sRGB-based default colors. <br>– **Dark mode variant:** Supported via `.dark` class or media (configurable). <br>– **Plugins needed** for certain features (e.g. forms, typography, line-clamp, container-queries in 3.3).	**New utilities in v4:** Many additions – e.g. **Text shadow** utilities (`text-shadow-*`) are finally included [29] [30] ; **Masking** utilities (`mask-*`) allow using images/gradients as masks [30] ; **3D transforms** like `rotate-x-*`, `rotate-y-*`, `translate-z-*` for 3D CSS transforms [10] ; **Colored drop-shadows** (`text-shadow-color` or opacity modifiers on shadows) [31] [32] ; **Fine-grained text wrapping** (`overflow-wrap` utilities for better text break control) [33] ; **Safe align** utilities for flex/grid (to avoid losing content on overflow); **Pointer query** variants (`pointer-*` and `any-pointer-*`) to target hover-capable vs touch devices [31] ; etc. <br>– **Color system:** Upgraded default palette to use OKLCH (P3 color space) for more vibrant colors (keeping names same) [34] . Also supports CSS `color-mix()` for transparency (e.g. `.bg-opacity` now done via `color-mix` internally) [14] [35] . <br>– **Dark mode:** Still supported (no fundamental change in usage), but new variants like `inverted-colors` (for forced light/dark modes) are added [36] . <br>– **First-party plugins updated:** Official plugins (forms, typography, etc.) are updated to align with v4's CSS-first approach and can leverage CSS variables and dynamic values [37] .

Aspect	**Tailwind CSS 3.x** (v3)	**Tailwind CSS 4.1** (v4.1.11)
Deprecated/ Removed	*– N/A (base reference)*	**Removed in v4:** JavaScript config is no longer auto-loaded (must use `@config` if needed) [5]. Dropped config options: `corePlugins` (cannot disable core utilities via config), `safelist`, and `separator` options are removed [38]. <br>– Removed old deprecated utility classes: e.g. `*-opacity-*` utilities for colors are gone (use new color opacity syntax like `bg-black/50` instead) [39]. Legacy aliases like `flex-grow-0`/`flex-shrink-0` are replaced by `grow-0`/`shrink-0` [39]. <br>– Renamed utilities for consistency: e.g. `shadow-sm` → `shadow-xs`, `shadow` (no suffix) → `shadow-sm`, likewise for `blur`, `drop-shadow`, `rounded` sizes [40]. `outline-none` from v3 is renamed `outline-hidden` (and a new true `outline-none` is introduced) [41] [42]. The base `ring` utility now equals a 1px ring (was 3px in v3), so old `ring` usage should become `ring-3` for same effect [43]. <br>– **CorePlugins disabling**: The `corePlugins` config array to turn off built-in utilities is no longer supported [44] (v4 expects you to simply not use what you don't need, and rely on tree-shaking). <br>– **Browser support:** v4 targets modern browsers (Safari ≥16.4, Chrome ≥111, Firefox ≥128) and won't work in older browsers due to use of modern CSS features like `@property` and `color-mix()` [45]. Developers needing IE11 or very old Safari support must stay on v3.4 [46].

(Sources: Tailwind CSS official v4.0 and v4.1 release notes [14] [29], Tailwind v3→v4 upgrade guide [47] [40], and community summaries [4] [48].)

Configuration and Theming Differences (tailwind.config.js vs CSS-first)

Tailwind 3.x: Traditional JS Config – In v3, almost all customization was done via the `tailwind.config.js` file. This JavaScript config defined the theme (colors, spacing scale, breakpoints, etc.), variants, plugins and content paths. The developer had to maintain this separate file and update it for any design token changes or to enable certain features [1]. For example, adding a new color or breakpoint meant editing the JS config. Also, to remove unused styles, one needed to list all template file paths in the `content` array of the config (or `purge` in older v2 terms) so Tailwind's JIT compiler knew where to scan for class usage [2]. In short, **v3 required a config file** for most projects, and manual setup for things like content scanning and theme extension.

Tailwind 4.1: CSS-First & Optional Config – Tailwind v4 introduces a *CSS-first configuration paradigm*, meaning you can now customize Tailwind directly within your CSS files instead of (or in addition to) a JS config file [3]. Key changes include:

- `@theme` **Directive for Design Tokens:** You can define theme values (colors, fonts, breakpoints, spacing scale, etc.) using CSS custom properties under an `@theme` at-rule in your CSS. Tailwind will pick up these CSS variables as your design tokens [3]. For example, instead of adding a color in `tailwind.config.js`, you could write:

```
@theme {
  --color-brand-primary: #3b82f6;
  --font-sans: "Inter", sans-serif;
  --spacing-lg: 2rem;
  --breakpoint-xl: 80rem;
}
```

This will extend/override Tailwind's defaults with your provided values (the variables get registered with a `--tw-` prefix in output) [49]. This keeps all styling concerns in CSS, aligning with modern CSS workflows [50]. *The JS config is no longer the primary source of truth for theme customization.*

- **No Auto-Loaded Config File:** Tailwind v4 does **not automatically load** a `tailwind.config.js` file if one exists. In fact, new projects don't even require a config file. If you do have a config file (for complex configurations or legacy reasons), you must explicitly import it in your CSS using an `@config` directive, e.g. `@config "tailwind.config.js";` [5]. This is a big shift from v3, where having a config file in the project root would be picked up by Tailwind automatically. The new approach encourages using CSS for config, making the config file opt-in.

- **Simplified Defaults and No-Config Needed:** Tailwind 4 works out-of-the-box with smart defaults. Common settings like breakpoints, container centering, default colors, etc., are pre-configured, so you often **don't need a config file at all** if you're fine with the defaults [51] [52]. In v3 one would usually initialize a config (via `npx tailwindcss init`) even if just to set content paths; in v4 you can skip this step for many cases [53]. The framework auto-detects source files for purging unused CSS (see Build section) and uses built-in theme values.

- **Removed/Changed Config Options:** Some config options from v3 are removed in v4, reinforcing the new approach:

- The `corePlugins` option (which let you disable specific built-in utilities) is **no longer supported** [38] [44]. In v3 you could do `corePlugins: { preflight: false }` or provide an array of utilities to disable. In v4, you can't toggle core plugins via config; you either override their styles via CSS or simply not use those classes (unused CSS will be dropped in the build). This change simplifies the build and encourages relying on tree-shaking for unused CSS.

- The `safelist` option (for whitelisting specific class names to always include) is removed [38]. This is replaced by a new on-demand mechanism: the `@source inline(...)` directive can be placed in CSS to include utilities that don't appear in your content files [54]. Essentially, safelisting is now done inline rather than in config.
- The `separator` option (which allowed customizing the character used to separate variant prefixes, default ":") is also dropped [38]. Tailwind 4 uses a consistent `:` separator and variant grouping now handles complex cases differently (see **Variants** below), so this option was deemed unnecessary.

- **Theme Extension via CSS Variables:** In v3, extending the theme (e.g., adding a new color or spacing value) was done by merging into the theme object in config. In v4, when using CSS-first, you can simply declare a new custom property in `@theme`. All theme values in v4 are exposed as CSS custom properties (prefixed with `--tw-`), which means you can even consume them in runtime CSS or other tools easily [55]. This also allows dynamic theming (like toggling themes by switching out CSS variables, without rebuilding Tailwind). For instance, dark mode can be implemented by adjusting `--color-*` variables under a `.dark` selector, etc.

- **Backward Compatible Config (with Catches):** You *can* still use a JS config file if needed – Tailwind 4.1 provides backward compatibility for teams not ready to fully migrate [56]. However, you must load it via `@config` as noted, and not all old options work. Tailwind's upgrade tooling can convert many config settings into the new CSS variable format automatically [57] [58]. If an AI or tool was reading a tailwind.config.js in v3, note that in v4 it might not exist or might require explicit inclusion. The AI should be prepared to parse theme values from CSS `@theme` sections as well.

Why this matters: For an AI system generating or editing Tailwind code, these config changes mean that it should consider outputting CSS-based customizations rather than always producing a JS config. It should also **avoid using deprecated config fields** (like `safelist` or `corePlugins`) in Tailwind 4 code [38]. Instead of writing logic that inserts values into a JS object, the AI might generate an `@theme` block in a CSS file when running in a Tailwind 4 context. Moreover, the AI should know that **class generation is more flexible** in v4 (no need to predefine every value in config), which reduces the need to modify config for new utilities (see next section).

Utility Class Generation Changes

One of the biggest developer-facing changes in Tailwind v4 is how utility classes are generated and what values they support. Changes in this area will affect both how you *write class names* in HTML and how Tailwind's engine produces CSS for them.

- **Dynamic Utility Values (Arbitrary values without `[]`):** In Tailwind 3, if you needed a utility with a value outside the default scale, you had two options: extend the theme config or use an *arbitrary value* (in square brackets) in your class. For example, if the default spacing scale went up to `p-64` and you needed `p-72`, you might do `p-[18rem]` in your HTML or add `72: '18rem'` in the theme. Tailwind 4 simplifies this: **many utilities now accept any numeric value by default**, without requiring the `[...]` arbitrary syntax or config entry [59] [60]. The framework internally uses formulas and CSS variables to calculate these values.

- *Example:* In v3, `grid-cols-15` (15 columns) would not work unless you extended the `gridTemplateColumns` scale in config (default max was 12). In v4, you can write `<div class="grid grid-cols-15">` and it will just work – Tailwind will generate the appropriate CSS (`grid-template-columns: repeat(15, minmax(0,1fr))`) [61] [9]. Similarly, classes like `w-103` (an arbitrary size) are allowed and translated into `width: calc(var(--spacing)*103)` if within the spacing context [8] [62]. Essentially, **any integer (or supported unit value) for sizes, spacing, z-index, columns, rows, etc., is accepted**. This "dynamic value" capability means developers no longer have to guess which values exist in the default scale or go define new ones for simple cases [63] [64].

- Under the hood, Tailwind v4 defines a single base spacing unit (`--spacing`) and multiplies it for any `*-(number)` class [62]. The default `--spacing` is 0.25rem (the step used in Tailwind's spacing scale), so a class like `.mt-8` becomes `margin-top: calc(var(--spacing) * 8)` [65]. This approach dramatically reduces the amount of generated CSS and makes extending to new values trivial. The v4 upgrade guide notes that if you had arbitrary values in v3 (e.g. `mt-[34px]`), the upgrade tool will convert them to the simpler v4 form if possible (e.g. `mt-8`) [66].

- **No More Guessing Scale Ranges:** Because of dynamic values, you don't have to remember if a utility exists. For instance, in v3 the largest text size was `text-9xl`. If you needed a `text-10xl`, you'd have to extend the fontSize scale. In v4, if you try `text-10xl` it will likely work out-of-box if it follows the established increment pattern (though for font sizes the defaults are still fixed steps). For straightforward numeric scales (spacing, sizing, grid, flex, order, z-index), **v4 treats the numeric part as dynamic**. The StaticMania comparison notes you can even use values like `w-103` or `z-40` directly where previously you might be constrained or need config [8] [67].

- **Arbitrary Properties:** Tailwind 3 introduced arbitrary values (e.g. `bg-[#123456]` for a custom color). Tailwind 4 goes further by allowing *arbitrary CSS properties* through a special class syntax. You can write classes like `style-[<property>:<value>]` to generate custom CSS on the fly [7]. For example, `style-[scroll-margin-top:2rem]` or the given example `style-[scroll-snap-align:start]` [7] will output exactly that CSS on the element. This means if a CSS property isn't (yet) supported by Tailwind's core utilities, you can still use it without writing a separate CSS file or plugin – a big boost for forward compatibility with new CSS features [68]. In v3, arbitrary values were limited to known utility categories (you could plug in custom *values* but not arbitrary *properties* except via adding custom CSS). In v4, this change effectively lets developers (or AI systems) express any CSS as a utility class string, making Tailwind a more complete superset of CSS.

- **New Utility Classes:** Tailwind 4.x added numerous utility classes that did not exist in v3 core. These new utilities expand what can be done without custom CSS:

- **Text Shadows:** After years of omission, v4.1 introduced `text-shadow` utilities with default sizes from `text-shadow-2xs` up to `text-shadow-lg` [30] [69]. This means you can add text shadows to text elements using classes, whereas in v3 you had to write custom CSS or use a plugin. You can also color the text shadow with classes like `text-shadow-red-500` or apply opacity modifiers (e.g. `text-shadow-lg/50` for 50% opacity shadow) [32] [70].
- **Masking Utilities:** Tailwind 4.1 added utilities for CSS masks (using `mask-image`). These include classes like `mask-[url(...)]` for image masks or built-in gradient masks such as `mask-radial-`

`*` variants for radial fade effects [30] [71]. V3 had no mask utilities; v4 now covers this modern CSS feature in core.

- **3D Transforms:** V4.0 introduced 3D transform utilities like `rotate-x-*`, `rotate-y-*`, `rotate-z-*`, `translate-z-*`, `perspective-*`, etc. [10] [72]. V3 only supported 2D transforms (rotate, translate, scale in X/Y). With v4, you can directly rotate elements in 3D space with classes (e.g., `rotate-x-45` to rotate 45° around the X-axis).
- **Container Query Utilities:** V4.0 integrated the container queries plugin into core. While not exactly "utilities" in the same sense (it uses the `@container` at-rule), you now have pseudo-utilities like the `@container` token and container-based variants (`@sm:`, `@lg:` etc. when inside a `@container`) for responsive designs based on parent size [73] [74]. V3 lacked this entirely (until a later plugin); v4 allows things like `<div class="@container ...">` and then inside it `<div class="bg-red-500 @md:bg-blue-500">` to change color when the container hits the "md" width, etc. This change affects how one can structure responsive classes (the AI should know `@sm:utility` is now valid inside a container context).
- **Miscellaneous New Utilities/Variants:** V4.1 added specialized variants and utilities, e.g., `pointer-hover:` or `any-pointer-coarse:` to target devices with or without hover capability [31], `noscript:` variant to style when `<noscript>` is active [36], form control state variants like `user-valid:` / `user-invalid:` for inputs [36], `inverted-colors:` to detect forced color schemes, new alignment utilities like `items-baseline-last` and `items-safe` for safe alignment in overflow situations [75], etc. All these did not exist in v3.

In summary, **Tailwind's utility coverage is much broader in 4.1**. An AI generating Tailwind code can leverage these new classes (like text shadows, masks, 3D transforms) instead of having to produce custom CSS or assume they're impossible. It should also be careful with renamed utilities (e.g., use `shadow-xs` instead of v3's `shadow-sm`) and with changed default behaviors (e.g., using `ring` alone now gives a 1px ring, not 3px [43]).

- **Utility Renames and Behavioral Changes:** Some existing utilities were tweaked for consistency:
- Sizing suffix changes: The smallest size suffix for shadows, blurs, etc., changed from `-sm` to `-xs` in v4, and the base name now corresponds to the next size up [76]. For example, in v3 `shadow-sm` was the smallest drop shadow; in v4 it's renamed `shadow-xs`, and what was `shadow` (no suffix) in v3 is now `shadow-sm` in v4 [76]. Similar for `blur-sm` → `blur-xs`, `rounded-sm` → `rounded-xs`, etc. This means an AI upgrading code should replace these class names accordingly. (Tailwind v4 still accepts the old names for backward compat, but they map to different actual styles [77], so it's safer to use the new naming.)
- Outline and Ring defaults: The `outline-none` utility in v3 did not remove outline entirely (it left an invisible outline for accessibility), which was confusing. V4 renames that old behavior to `outline-hidden` and introduces a true `outline-none` that sets `outline: none` [42]. If the AI is adding `outline-none` to remove focus outlines, it must be aware that in v4 this *will* remove it (and maybe consider using `outline-hidden` if the intention was only to hide it visually but keep it in forced-color mode). Also, the base `ring` class (for focus rings) changed from a 3px default width to 1px, and its default color from blue to `currentColor` [43] [78]. So, code that relies on `ring` without a size or color in v4 will behave differently than in v3. The recommendation is to always specify the ring size and color in v4 (or set theme variables to preserve old behavior) [79] [80].
- Space-between utilities (`space-x-*` / `space-y-*`) now use a different selector under the hood for performance: instead of selecting `:not([hidden]) ~ :not([hidden])`, they use `:not(:last-child)` to apply margin-bottom or margin-right [81]. Functionally this is similar, but it

may differ in edge cases (like inline elements or when combining with other margins) [82]. Generally this shouldn’t require user changes except in rare scenarios, but it's good to note that the behavior of spacing between elements is slightly adjusted in v4.

- Gradients: Tailwind 4 renamed `bg-gradient-to-*` utilities to `bg-linear-to-*` (to make way for `bg-radial-*` and `bg-conic-*`) [83]. Also, overriding gradient stops with variants works more intuitively in v4 – it preserves other stops instead of resetting them [84] [85]. For instance, in v3 if you did `dark:from-blue-500` on a gradient that had `to-yellow-400`, the `to` color might reset to default (transparent) unexpectedly; v4 fixes this so only the intended stop changes [84] [85].
- **Hover on touch devices:** Not exactly a class name change, but in v4 the `hover:` variant CSS is wrapped in a `@media (hover: hover)` media query [86]. This means on touch devices (which usually report `hover: none`), `:hover` styles won’t apply on tap by default. In v3, hover styles would apply on touch (typically on first tap). This might affect interactive behavior; the upgrade guide notes that if your site relied on hover triggering on mobile taps, you might need a custom variant to revert to the old behavior [24] [87]. For an AI, this means acknowledging that `.hover:utility` in v4 is truly meant for hover-capable devices only – this aligns with modern best practices.

Implications: With v4’s flexible class generation, an AI system can be more free-form in suggesting Tailwind classes: - It can directly suggest classes for arbitrary numeric values (like width, spacing, columns) without instructing the user to edit a config. - It can utilize new classes for effects that previously required custom CSS (text-shadow, mask, 3D, etc.), resulting in more idiomatic Tailwind code. - It should update any old class names or usage patterns to the new ones (e.g., avoid now-removed classes, use correct renamed classes, apply `@container` where needed instead of older responsive hacks). - The system should also remember that some utilities now rely on CSS variables – for example, colors are implemented via `--tw-color-<name>` variables and transparency via `color-mix()` [14] [35] – but typically this does not change how you *write* the class, only how the output works. One noteworthy change: using CSS custom properties in arbitrary values now requires parentheses instead of square brackets (due to changes in CSS spec) – e.g., in v3 you could do `bg-[--my-color]`, in v4 it must be `bg-(--my-color)` [88].

In summary, Tailwind 4’s utility system is more powerful and less config-bound than 3’s. The AI should leverage these improvements for more concise and capability-rich class generation, while adjusting any outdated class names to their v4 equivalents.

Build Process & JIT Compilation Differences

Tailwind CSS 3.x made a big shift by introducing JIT (Just-In-Time) compilation, generating CSS on the fly based on usage. Tailwind 4.x takes this further, with internal engine improvements and workflow streamlining that affect how projects are set up and built:

- **Always-On JIT and Better Performance:** Tailwind 3’s JIT was often enabled by default, but it still required scanning specified content files and could incur overhead for initial builds. Tailwind 4 is a **ground-up rewrite focused on performance** [89]. The result is dramatically faster build times:
- Full builds up to ~5× faster, and **incremental builds 8–100× faster** in v4 compared to v3, according to Tailwind Labs’ benchmarks [14] [15]. Especially, when no new classes are added (i.e., just editing HTML that uses existing classes), rebuilds complete in microseconds in v4 [90]. This speed-up comes from an optimized engine and using newer JS runtime features.

- The JIT process is always on and cannot be turned off (v3 had effectively already made it default). In v4, there is no separate "purge" stage – unused CSS is *always removed* automatically. The Medium overview succinctly notes: "Tailwind v4 automatically removes unused CSS based on the content array. No extra setup—just smaller, faster builds." [13]. In other words, **tree-shaking is built-in**. As long as Tailwind knows where your templates are, it will include only the classes it finds. If you don't configure any content paths, v4 attempts an automatic content discovery.

- **Automatic Content Detection:** In Tailwind 3, you *had to specify* your HTML/JSX template paths in the config for Tailwind to scan. Forgetting to do so could mean missing styles or, conversely, including everything (if you left purge disabled). Tailwind 4 introduces **automatic content source detection** [91]. This means if you do not provide a content list, Tailwind will try to intelligently find your project's files that could contain class names. It likely uses sane defaults (e.g., all `.html`, `.js`, `.jsx`, `.ts(x)`, `.vue`, etc., excluding `node_modules` and files listed in .gitignore) [92]. This lowers the entry barrier – new users can install Tailwind and start coding without even writing a config file, and still get optimized builds. However, for unusual project structures, you might still manually specify content. For an AI, this means that when instructing how to set up Tailwind v4, mentioning content paths is optional (whereas for v3 it was mandatory to avoid a huge CSS file). Additionally, v4 adds directives to fine-tune content scanning in CSS:

- `@source not <paths>` can exclude certain large folders (like heavy libraries) from scanning to speed it up [93].

- `@source inline([...])` can explicitly safelist classes as mentioned (which replaces config.safelist).

- **Simplified Build Configuration:** Tailwind 4 reduces external dependencies and config in the build toolchain:

- **PostCSS Plugin Changes:** In v3, Tailwind's npm package itself was a PostCSS plugin. You would typically use it alongside `postcss-import` (to handle `@import` rules in your CSS) and `autoprefixer` (to add vendor prefixes). In v4, Tailwind has separated concerns:
    - The Tailwind package (`tailwindcss`) now primarily provides the JIT engine and CLI. If you are using PostCSS, you should use the `@tailwindcss/postcss` plugin which wraps Tailwind and includes automatic import handling and autoprefixing built-in [16]. Essentially, `@tailwindcss/postcss` replaces the combination of `postcss-import + tailwindcss + autoprefixer` into one. The upgrade guide even says you can remove `postcss-import` and `autoprefixer` from your setup when migrating to v4 [16] [94].
    - If using **Tailwind CLI**, note that the CLI is now a separate package: `@tailwindcss/cli` [18]. In v3 you could run `npx tailwindcss ...` directly after installing `tailwindcss`. In v4, after installing, you might run `npx @tailwindcss/cli ...`. The CLI usage is otherwise similar, but splitting it helps lighten the core package and allow independent updates.
    - For **Vite** users: Tailwind 4 provides a first-party Vite plugin `@tailwindcss/vite` [17]. Instead of going through PostCSS, you can integrate it in `vite.config.js` as a plugin (which likely wraps the same functionality but optimized for Vite's dev server and HMR). The upgrade guide suggests migrating to this for better performance if you use Vite [95].

    - For other build tools (Webpack, etc.), the usage remains adding Tailwind via PostCSS, but with the new plugin approach.
- **CSS Import vs Directives:** As mentioned, you no longer use `@tailwind base; @tailwind components; @tailwind utilities;` in your main CSS. Instead, you add **one line**: `@import "tailwindcss";` [19]. This import triggers Tailwind to inject its layers (base, components, utilities) automatically. It simplifies setup and avoids potential ordering issues. (Tailwind still supports layering via cascade layers internally, but the user doesn't manage three separate directives anymore.)

- **No Extra Prefixer/Polyfill Steps:** Tailwind 4 uses Lightning CSS under the hood for things like autoprefixing and transforming modern CSS syntax for older browsers where possible [96]. This means when Tailwind builds your CSS, it already handles vendor prefixes (for things that need them, like certain flexbox or grid properties) and allows you to use modern CSS (like nesting, or new color functions) without additional tooling. In contrast, v3 needed autoprefixer and did not natively handle features like CSS nesting.

- **Modern CSS Features in Build:** Tailwind 4's engine leverages new CSS capabilities:

- It uses **CSS Cascade Layers** (`@layer`) natively to manage the ordering of base, components, utilities, and any overrides [14] [35]. V3 had a custom layering system; v4 aligns with the CSS spec's layering, which reduces specificity issues and gives more control.
- It registers **CSS custom properties with** `@property` for certain dynamic features (like transitions for gradients, etc.) [14] [35]. This is why v4 needs modern browser support – it declares custom properties with types so they can be animated (e.g., `--tw-gradient-from` is registered so it can transition colors smoothly [14] [35]).
- V4 outputs logical properties (for RTL) and uses functions like `color-mix()` for color transparency. For example, a class like `.bg-blue-500/50` (blue with 50% opacity) in v3 might compile to `background-color: rgba(..., 0.5)`, whereas in v4 it compiles to `background-color: color-mix(in oklab, var(--color-blue-500) 50%, transparent)` [14] [97]. This yields better fidelity especially with the OKLCH color space.
- **Native CSS nesting** is allowed in your input CSS if you use `@apply` or write custom styles. Since Tailwind's build uses Lightning CSS or similar, it can process the `&` nested selectors into proper CSS for output [26]. V3 did not support this natively (you'd need to add a separate PostCSS nesting plugin).

Implications for AI/system: - The AI should simplify setup instructions for Tailwind 4: e.g., "install `tailwindcss` and `@tailwindcss/postcss` (or `@tailwindcss/cli`), then put `@import "tailwindcss";` in your CSS" [98] – instead of the longer v3 process of adding multiple directives and plugins. It should also caution that older setups with autoprefixer can be slimmed down [16]. - When upgrading or generating code, the AI should ensure that it references the correct build commands or packages (using the new CLI package, using the vite plugin if applicable, etc., rather than outdated commands). - Because v4 builds are much faster, iterative class generation (which an AI might do interactively) is more feasible. The AI can encourage using the latest version for performance reasons. - The AI should note that if someone is on an older environment (older Node or older browser targets), they might need to stick with v3.4 due to compatibility. Tailwind 4 requires relatively modern browsers and Node (the upgrade tool needs Node 20+, though Tailwind itself likely runs on Node 16+). - Finally, the improved tree-shaking means an AI doesn't have to overly warn about large file sizes if content is properly detected. In v3, forgetting to configure purge could lead to multi-MB CSS; in v4, that scenario is much less likely thanks to automatic content detection. But if the AI is generating a config file for some reason, it should omit the deprecated purge keys and either rely on auto or use the `content` key (still supported) with globs.

New Syntax & Usage Paradigms in Tailwind 4.x

Tailwind 4 not only changes *what* you can do, but also *how* you express certain things. Several new at-rules and syntax conventions were introduced, altering how developers write Tailwind-powered CSS:

- **Using `@import` instead of `@tailwind` Directives:** As noted, you now include Tailwind in your CSS via a single import. In Tailwind 3, a typical `styles.css` might start with:

  ```
  @tailwind base;
  @tailwind components;
  @tailwind utilities;
  ```

  In Tailwind 4, you simply write:

  ```
  @import "tailwindcss";
  ```

  This one line brings in all of Tailwind's base, components, and utilities layers [19]. The change is mostly for simplicity – functionally, the result is similar – but it means any AI or code mod should update the inclusion method. **The old `@tailwind` at-rules are removed** and will error if used in v4 [47].

- **New `@theme` At-Rule:** As discussed in the config section, `@theme` allows defining custom theme values (design tokens) directly in CSS [3]. It essentially replaces what you would have put under `theme: { ... }` in a JS config. This new syntax means your CSS file can contain configuration. For example:

  ```
  @import "tailwindcss";
  @theme {
    --spacing-14: 3.5rem;
    --color-brand-500: #123456;
  }
  ```

  This would add a new spacing value and override a color. The presence of `@theme` in CSS is completely new in v4; an AI should generate it when appropriate (e.g., when user wants to add a new color) instead of modifying a JS file. Under the hood, Tailwind collects these and generates CSS variables (with `--tw-` prefix in :root) for them [49]. **All default theme values are now accessible as CSS vars** (e.g., `--tw-color-red-500` exists) [99], which even allows dynamic adjustments or usage in arbitrary values.

- **New `@utility` At-Rule:** In v3, if you wanted to add a custom utility class (say a class for a CSS property Tailwind didn't support), you had two main options: create a plugin in JS, or just write the CSS for that class. Many devs would add custom CSS inside `@layer utilities { ... }` so that it would be merged into Tailwind's output. In v4, since `@layer` is no longer hijacked by Tailwind (it respects actual cascade layers), the recommended way is to use the new `@utility` at-rule [23]. For example:

  ```
  @utility no-tap-highlight {
    -webkit-tap-highlight-color: transparent;
  }
  ```

  This declares a utility class `.no-tap-highlight` with the given styles. Tailwind will process it and include it alongside other utilities. The benefit of `@utility` is that Tailwind is aware of these classes and will automatically allow variant prefixes (hover:, sm:, etc.) on them [100] [101]. In v3, a class you defined in `@layer utilities` mostly worked with variants, but the ordering and specificity could be tricky. Now, `@utility` classes are sorted by the number of properties to ensure consistency (so "component"-like utilities with more styles yield to smaller utilities) [102]. For an AI, using `@utility` provides a cleaner way to add bespoke classes.

- **Custom Variants via `@variant` or `@custom-variant`:** Tailwind 4 opens up defining custom variants in pure CSS. In v3, adding a new variant (like a custom media query or state) required writing a plugin in JS (using the complex API of `addVariant` with a function). In v4, you can write:

  ```
  @variant focus-visible (&:focus-visible);
  ```

  to define a new variant that can be used as `focus-visible:` in your HTML classes, with the given selector expansion [22]. Tailwind also provides some built-in custom variants; e.g., the upgrade guide shows using `@custom-variant hover (&:hover)` to override how hover is implemented [24]. The difference between `@variant` and `@custom-variant` seems to be that `@variant` registers a new *named* variant (e.g., you give it a name before the selector), whereas `@custom-variant` might be used to override existing ones or define ad-hoc. The syntax essentially allows writing CSS selectors in a more direct way. For example, the default `dark:` variant (for class strategy) could be expressed as:

  ```
  @variant dark (&:is(.dark &));
  ```

  (The Medium article snippet shows `@variant dark (&:is(.dark *));` which is conceptually similar [22].)

Additionally, **Arbitrary variant groups** were introduced (in v4.1) which let you apply multiple variant conditions to a group of classes in one go. The syntax seen is:

```
<div class="[&:hover,&:focus]:bg-blue-500">Hover or Focus me</div>
```

This uses an arbitrary selector (the `[...]` syntax) containing multiple selectors separated by a comma [25]. It results in the background color applying on hover or focus without writing two classes. This is new to v4.1; v3 had no direct equivalent (you'd have to repeat the class with both `hover:` and `focus:`). For an AI dealing with Tailwind HTML, this means it can output more concise class attributes when multiple variants apply the same styles, improving readability.

- **CSS Nesting:** While not a Tailwind-specific syntax, it's worth noting that Tailwind 4 supports CSS nesting out of the box when writing custom CSS or applying utilities. For instance, if you have a component CSS file, you can do:

  ```
  .card {
    @apply bg-white rounded-lg;
    &-header { @apply text-lg font-bold; }
    &-body { @apply p-4; }
  }
  ```

  This will compile properly (the `&-header` becomes `.card-header` selector) [26]. Tailwind 3 would have required a separate nesting plugin or manual writing of the full selector. The integration of Lightning CSS means your nested CSS with `@apply` works seamlessly. An AI should be aware that writing nested classes with `&` is acceptable in Tailwind 4 environments (no need for workarounds).

- **Typed Class Support / IntelliSense:** Tailwind's 4.x also introduced official type definitions to improve editor support. Specifically, Tailwind 4 has **built-in TypeScript support** for config files [103]. If you use a `tailwind.config.ts`, the types for theme keys, etc., are available, improving autocomplete. More directly relevant to an AI, Tailwind's VSCode IntelliSense plugin uses the JIT engine to provide class suggestions as you type. In v4, this remains or improves, but now that classes can be dynamic (any number), the range of suggestions might effectively be infinite. The AI might lean on the fact that in v4, if a class pattern makes sense (like `grid-cols-N`), it's probably valid, rather than second-guessing if N is allowed.

Implications: The new syntax elements mean an AI needs to adapt how it outputs Tailwind-related code: - It should output `@import "tailwindcss";` for new projects instead of the three `@tailwind` lines [19]. - When guiding to add custom styling or theming, it should use `@theme` for v4. For example, "to add a new breakpoint in Tailwind 4, do `@theme { --breakpoint-<name>: <size>; }`" instead of editing a JS config. The Medium article even shows resetting default breakpoints via CSS by setting `--breakpoint-*: initial` then adding custom ones [104]. - For custom utility classes or variants, the AI can suggest writing them in CSS with `@utility` and `@variant`, rather than writing a JS plugin function as in v3. This lowers complexity. - The support for nesting and arbitrary variants can help produce cleaner code; the AI might generate grouped variant syntax to reduce duplication. - If migrating code from v3, the AI should remove or update the old syntaxes (like `@tailwind` directives, `layer` usage, etc.) to the new ones (which the official upgrade tool also does automatically [57]). - Also, since `separator` is gone, the AI should not try to use unusual variant separators or assume they can be changed. In v4, variant naming conventions (like using `:`) are fixed.

Overall, Tailwind 4's new syntax aligns the framework closer to pure CSS patterns (using imports, custom properties, cascade layers, etc.), which should make it easier for tools to reason about. An AI can treat many Tailwind-specific constructions as just CSS with certain patterns.

Removed and Deprecated Features in v4

Tailwind CSS v4 did introduce breaking changes, though the team tried to minimize pain by providing an upgrade tool. Key features or patterns from v3 that were removed or changed in v4 include:

- `@tailwind` **Directives:** As mentioned, the `@tailwind base/components/utilities` directives are **removed**. If an AI sees these in code targeted for v4, it should replace them with the single import line [19].

- **Old Utility Classes:** Any utility class that was marked as deprecated in v2 or v3 is likely gone in v4. For example:

- Color opacity utilities like `bg-opacity-50` or `text-opacity-75` are removed; in v3 they were already replaced by the slash notation (e.g. `bg-red-500/50` for 50% opacity) [39].
- Similarly, `divide-opacity-*`, `ring-opacity-*`, `placeholder-opacity-*` are removed [39].
- `flex-grow-0` / `flex-shrink-0` (old names) removed in favor of `grow-0` / `shrink-0` (these shorter aliases existed in v3 and now are the primary names) [105].
- `overflow-ellipsis` (an old v2 utility) was replaced by `text-ellipsis` in v3 and the old one is removed in v4 [105].
- `decoration-slice` / `decoration-clone` were renamed to `box-decoration-slice/clone` long ago; the old names are purged [106].

- These removals usually have direct replacements, so an AI should map them if encountered (the upgrade guide provided a list of deprecated → replacement [39]).

- **Default Config Options:** Some default behaviors changed as noted:

- The default **border color** (if none specified) in v3 was `gray-200` for borders/dividers. In v4 it's `currentColor` [107]. So `<div class="border">` will have a faint gray border in v3, but in v4 it will inherit the text color as border color. It's more CSS-native but could change appearance. The AI should encourage explicitly setting border color for consistency (which is good practice anyway) [108].
- The default **ring color** changed from `blue-500` to `currentColor` (and width 1px as mentioned) [78]. So focused elements using just `ring` get a subtler focus style unless a color is added.

- **Preflight adjustments:** v4's Preflight (base styles) has a couple changes:

    - Form inputs' placeholder color default is now 50% of current text color (instead of a fixed gray) [109] [110].

    - Buttons default `cursor` is now `default` (arrow) instead of `pointer` in v3's preflight [111]. This aligns with browser defaults (buttons normally don't show a hand cursor unless they are links). If code relied on the hand cursor, you'd need to add `cursor-pointer` explicitly in v4.
    - `<dialog>` margins no longer default to auto (centered) [112], which could affect any usage of the `<dialog>` element.
    - These are minor but notable if the AI is analyzing base style differences. They reflect Tailwind becoming less opinionated by default.

- **Dropped Browser Support:** As a breaking change, Tailwind 4 targets only modern browsers. If an AI tool is generating code meant for IE11 or legacy browsers, it must stick to Tailwind 3.4. Tailwind 4's reliance on modern CSS like `@property` (for custom prop definitions) and cascade layers means it will not function on older browsers [45]. The upgrade guide explicitly says to remain on v3.4 if older browser support is needed [46]. There's mention of a possible future "compatibility mode," but as of 4.1 no such mode is released [113].

- **Safelist and corePlugins Config:** Already discussed – these config options are removed. The AI should not generate a `safelist` in a config for v4 (instead use `@source inline`) or `corePlugins: []` (no longer works). Also, `experimental` flags from v3 (like `experimentalApplyComplexClasses` or others) are likely removed or always-on in v4.

- **Variant Ordering:** One subtle change was the **order in which multiple variants are applied** if written together. In v3, writing `sm:hover:text-red-500` versus `hover:sm:text-red-500` could produce different outcomes because of right-to-left processing. In v4, the parsing is left-to-right, meaning `sm:hover:color` is equivalent to `@media(min-width:sm) { &:hover{ color } }`. In practice, most people wrote variants left-to-right already (e.g. `dark:hover:`), so v4 making it left-to-right is more intuitive. The upgrade guide suggests there are few cases to change (mostly if using the `*` direct child variant combined with others) [20]. For the AI, it means it can assume the variants apply in the order written, which aligns with natural language order (no need to reverse them internally). If analyzing old code, though, it should be cautious that `first:hover:` in v3 might need flipping to `hover:first:` in v4 or similar in edge cases.

In summary, v4 pruned legacy cruft and aligned defaults with modern CSS. An AI system must ensure it doesn't use any of the removed utilities or config patterns when targeting Tailwind 4, and it should adjust any assumptions about default behaviors (like border color or ring thickness). Most of these changes aim to simplify things (less to configure, more consistency), so while they are "breaking," they usually have straightforward fixes.

Developer Ergonomics and Tooling Enhancements

Finally, Tailwind CSS 4.x introduced improvements that make life easier for developers (and AI assistants) working with Tailwind:

- **TypeScript & IDE Support:** Tailwind 4 has official **TypeScript support for configuration** [103]. You can write your config in TypeScript (`tailwind.config.ts`) and Tailwind provides type definitions so that your editor can autocomplete keys like theme names, etc. This reduces errors and improves DX. Even though v4 moves away from heavy config, for any remaining config or for plugin authors, having types is a boon. For an AI that might be interfacing with a codebase, knowledge that tailwind config can be in TS is useful (e.g., it should update file references accordingly).

Additionally, the **VS Code Tailwind IntelliSense** likely improved with v4's features. It can now suggest arbitrary properties and new classes. The AI could leverage the fact that any class it suggests will either be recognized by the IntelliSense (if valid) or flagged if not – meaning the space of valid classes is more dynamic but still checkable.

- **Improved Plugin API:** Tailwind's plugin system was refactored. While a lot of plugin usage shifts to CSS (with @utility, @variant), for more complex plugins (written in JS), Tailwind 4 introduced a new API that better aligns with the CSS-first approach. For example, plugins can tap into the new concept of CSS variables and design tokens. The StaticMania article notes that many community plugins were updated to use CSS variables and dynamic values, making them more powerful and easier to integrate [114]. Also, because all theme values are available as CSS variables, plugins can simply output CSS that references `--tw-*` variables, rather than needing to inject values via JS. For an AI, if it's generating a custom Tailwind plugin or instructing how to extend Tailwind, it should prefer the simpler routes (generate CSS with @utility or advise using official updated plugins) rather than complex JS manipulation.

- **First-Party Tools:** The introduction of a dedicated **Vite plugin** and separation of CLI means better integration into modern build workflows (Vite, Next.js, etc.). The AI can detect if the project uses Vite and suggest using `@tailwindcss/vite` for optimal performance [95], which reduces the need for manual configuration. Similarly, for PostCSS setups, use `@tailwindcss/postcss` to cut down config lines [16]. These official solutions improve developer experience by reducing boilerplate and potential misconfiguration.

- **Zero-Config and Defaults:** Tailwind 4's philosophy of working with zero config unless needed means new projects start faster. Developer ergonomics are improved by:

- Not requiring a config file (one less file to maintain for simple projects).
- Automatic template scanning means one less thing to think about when adding new file paths.

- The defaults have been tuned (e.g., color palette updated to look better on modern displays [34], more sensible base styles like no default pointer cursor on buttons [111], etc.), so developers (and by extension AIs) can rely on out-of-the-box styles being closer to expectations without immediate tweaking.

- **Migration Aids:** Though not a long-term feature, it's worth noting the Tailwind team provided an **upgrade tool (`npx @tailwindcss/upgrade`)** that automates many changes [115]. This is part of developer experience because it shows a commitment to smoothing breaking changes. The AI could incorporate knowledge of this tool if asked about upgrading – it does things like rename classes (shadow-sm→shadow-xs) and convert config to @theme, etc., automatically [116] [117]. This indicates that many repetitive changes are well-defined.

- **Better Error Messages:** Although not explicitly documented in our sources, major version updates often include improved error or warning messages to guide developers (e.g., if you accidentally use a removed class or old directive, Tailwind might throw a clear error). This indirectly helps because the AI could anticipate those errors.

- **Safety and Specificity:** Tailwind 4 uses Cascade Layers which means if a developer (or AI) writes custom CSS without thinking of order, Tailwind's layers ensure utilities typically win unless overridden intentionally. This reduces unexpected "why isn't my class applying" issues. Also, the sorting of custom `@utility` by number of properties means single-property utilities (like most built-ins) have higher priority than multi-property "components" [102]. For instance, a `.btn` class defined with multiple styles will yield to a `.text-red-500` utility on the same element, making utilities reliably override broader styles (unless you intentionally change layering). This consistency makes it easier to predict results.

Implications: The main takeaway for an AI system is that **Tailwind 4 is more developer-friendly and aligned with web standards**, which might simplify the logic needed to generate correct Tailwind code: - The AI should produce outputs that leverage defaults and require less configuration overhead. - When generating custom styles, lean on CSS (perhaps even generating a `<style>` block with `@tailwindcss` import and `@theme` definitions if needed) rather than instructing low-level config tweaking. - Recognize that with TypeScript support, when editing or creating config files, using `.ts` is acceptable and types (like `Theme['spacing']`) can be leveraged. - Since a lot of values are exposed via CSS variables, the AI could even suggest dynamic runtime adjustments (like "you can toggle themes by changing `--tw-color-primary` in CSS") which wasn't as straightforward in v3.

Finally, from an AI adaptation perspective: if the AI was trained predominantly on Tailwind 3 or earlier patterns, it should update its knowledge base with these differences: - No more default config file assumption. - Class names changes (no outdated names). - New abilities (so it doesn't wrongly say "Tailwind can't do X" if X is now possible). - Embrace `@theme`/`@utility` usage in answers or code generation for Tailwind 4 projects. - Ensure any coding style aligns with v4 conventions (for example, using logical properties like `ms-`/`me-` for margin start/end if needed, as v4 uses margin-inline under the hood for `mx` etc., though the API is same).

Conclusion: Tailwind CSS 4.1.11 vs 3.x presents a significant shift to a more modern, flexible, and performant framework. For developers and AI systems dealing with Tailwind: - **Utility API** is broader and more fluid in v4 (dynamic values, more built-ins). - **Configuration** is simpler (often moving from JS to CSS, or not needed at all). - **Build process** is faster and easier to configure (less config files, automatic optimizations). - **Syntax** has evolved to be more CSS-like and powerful (imports, nesting, grouping). - **Dropped features** clean up legacy baggage (which means updated practices for usage).

Adapting to these changes will allow an AI that generates or modifies Tailwind code to produce more accurate, up-to-date, and efficient outputs, taking full advantage of Tailwind 4's capabilities while avoiding pitfalls from outdated habits. Each of the differences outlined above should be accounted for in the AI's logic to ensure compatibility with Tailwind CSS 4.1.11 and an optimal developer experience.

Sources:

- Tailwind CSS v4.0 Release Notes – Tailwind Labs (Adam Wathan) [14] [91] [60] [27]
- Tailwind CSS v4.1 Release Notes – Tailwind Labs (Adam Wathan) [30] [31] [118]

• Official Tailwind CSS v3→v4 Upgrade Guide [47] [40] [119] [5]
• *Tailwind 4 vs Tailwind 3: Key Differences* – StaticMania (Md. Saad) [4] [2] [48]
• *What's New in Tailwind CSS v4?* – Medium (Mulugeta A. Gobeze) [13] [25] [22]
• Tailwind CSS Documentation (v4.1) – various sections on utilities and core concepts [43] [23] .

---

[1] [2] [3] [4] [6] [7] [8] [12] [37] [48] [51] [52] [53] [59] [67] [68] [92] [114] Tailwind 4 vs Tailwind 3: Key Differences and Improvements
https://staticmania.com/blog/tailwind-v4-vs-v3-comparison

[5] [38] [50] [56] [104] Tailwind CSS 4.1: Say Goodbye to Config Files, Hello to CSS-First Configuration | by Mohammed Aqib | Jun, 2025 | Medium
https://medium.com/@aqib-2/tailwind-css-4-1-say-goodbye-to-config-files-hello-to-css-first-configuration-a068b3a25c76

[9] [10] [11] [14] [15] [17] [21] [27] [28] [34] [35] [55] [60] [61] [62] [63] [64] [65] [66] [72] [73] [74] [83] [89] [90] [91] [96] [97] [98] [117] Tailwind CSS v4.0 - Tailwind CSS
https://tailwindcss.com/blog/tailwindcss-v4

[13] [22] [25] [26] [103] What's New in Tailwind CSS v4? A Quick Overview | by Mulugeta Adamu Gobeze | Medium
https://medium.com/@mulugeta.adamu97/whats-new-in-tailwind-css-v4-a-quick-overview-6bef483b3942

[16] [18] [19] [20] [23] [24] [39] [40] [41] [42] [43] [44] [45] [46] [47] [49] [57] [58] [76] [77] [78] [79] [80] [81] [82] [84] [85] [86] [87] [88] [94] [95] [99] [100] [101] [102] [105] [106] [107] [108] [109] [110] [111] [112] [113] [115] [116] [119] Upgrade guide - Getting started - Tailwind CSS
https://tailwindcss.com/docs/upgrade-guide

[29] [30] [31] [32] [33] [36] [54] [69] [70] [71] [75] [93] [118] Tailwind CSS v4.1: Text shadows, masks, and tons more - Tailwind CSS
https://tailwindcss.com/blog/tailwindcss-v4-1