ChatGPT

# Advanced Neovim Configurations Based on LazyVim (2025–2026)

Below are several popular and actively-maintained Neovim configurations built on the **LazyVim** framework (a pre-configured Neovim setup). These configs emphasize advanced features like AI-assisted coding, cross-platform use (desktop and even mobile/remote setups), modular plugin management, and reproducible setup (e.g. NixOS support), all while leveraging LazyVim's plugin ecosystem.

## Folke's LazyVim Dotfiles (folke/dot)

- **GitHub:** folke/dot – "My Dot Files" (☆ ~1.2k)
- **Description:** Personal Neovim configuration of Folke Lemaitre (author of LazyVim). Folke's dotfiles are *built on the LazyVim distribution* for a highly polished, IDE-like experience [1]. This config is widely used as a reference by the community due to its clean structure and up-to-date best practices.
- **Notable Plugins:** Includes all LazyVim default plugins (treesitter, LSP, telescope, etc.) plus Folke's own creations (e.g. **which-key**, **tokyonight** theme, **trouble.nvim**, **noice.nvim**). AI/code-assist tools (like GitHub Copilot) can be added easily – Folke's setup even inspired others to integrate LazyVim for AI-assisted workflows [2] [1].
- **Platform Compatibility:** Primarily used on Linux (Folke uses Hyprland on NixOS). However, it relies on LazyVim's cross-platform defaults, so it works on Mac and Windows (WSL) as well. Folke's config can be installed via Ansible scripts in the repo, making setup on new machines reproducible.
- **LazyVim Integration: Native LazyVim base** – the config imports LazyVim as the core, then overrides/extends it via custom Lua modules. This means Folke's dotfiles stay compatible with upstream LazyVim updates while adding personal tweaks [1]. Users can study this as a canonical example of extending LazyVim without forking it.
- **Unique Features:** Highly polished UX with minimal custom code – Folke's philosophy is to *leverage community plugins* rather than reinvent. The config uses **LazyVim's extras** (for example, enabling optional features like a Nushell LSP or others as needed) and showcases advanced LazyVim usage (e.g. using **Lazy** commands to update plugins, custom theme integration, etc.). It's effectively a **reference LazyVim power-user setup**, maintained by LazyVim's creator.

## jellydn's LazyVim IDE Template (jellydn/lazy-nvim-ide)

- **GitHub:** jellydn/lazy-nvim-ide (☆ 142)
- **Description:** A ready-to-go Neovim IDE configuration built on LazyVim. This repository started from the official LazyVim starter template and was *extensively enhanced* for full IDE functionality [3]. It comes pre-configured for web development out of the box (JavaScript/TypeScript, React, Tailwind CSS, etc.) and can be used as a template for your own config.
- **Notable Plugins: LSP** servers and **nvim-treesitter** parsers for web languages are pre-set. It includes a rich set of plugins that LazyVim provides (e.g. Telescope, Treesitter, Lualine, Gitsigns) plus extras like **ESLint/Prettier integration** and markdown preview. For AI-assisted coding, this config supports

adding tools like **Tabnine or Copilot Chat** – the author's guides show integration of Copilot Chat in Neovim [4].

- **Platform Compatibility:** Designed to work across OSes. The README provides a one-command installer and even a **Docker** example for trying it on any system [5]. It has tips for macOS (fixing key repeat rates) and should work on Linux (tested on Alpine via Docker). Windows is not explicitly documented here, but since it's based on LazyVim's cross-platform framework, it can run on Windows (or WSL) with minor adjustments (e.g. using the Windows install method from LazyVim).
- **LazyVim Integration: Full LazyVim distribution** – this config is essentially LazyVim plus additional plugins and settings. It tracks upstream LazyVim; you clone the repo and run `:Lazy sync` to get all plugins installed. The project is even marked as a "template" on GitHub, showing it's meant to be a starting point generated from LazyVim's starter [6].
- **Unique Features: Beginner-friendly IDE setup** – you get an IDE-like Neovim without manual setup. Notably, jellydn's config provides **demo projects and video tutorials** (links in README) for features like GitHub Copilot Chat, Gen.nvim (AI code generation), etc., all working within LazyVim [4]. It's also configured to be easily toggled or uninstalled (just remove the `~/.config/nvim` folder), making it safe to experiment with. This is a great choice if you want a *LazyVim-based config focused on web development and easy extensibility* [7].

## abzcoding/nv – AI-Enhanced LazyVim Configuration

- **GitHub:** abzcoding/nv (☆ 12, created late 2025)
- **Description:** A modern, feature-rich Neovim config built on LazyVim, with heavy emphasis on **AI-assisted workflows and performance optimizations** [8]. Despite its recent creation, it has gained attention for integrating cutting-edge plugins (including a local coding AI) into the LazyVim ecosystem.
- **Notable Plugins:** This config's standout feature is AI integration. It includes **Avante.nvim** (an open-source AI code assistant similar to Cursor IDE) and **GitHub Copilot** for intelligent code completion [8]. It also uses **Blink.cmp** (a Rust-based fast completion engine) for ⚡ snappy auto-complete [8]. Core dev tools are well-covered: robust **LSP** setup with enhanced diagnostics (via a tiny-inline-diagnostic plugin) [9], **treesitter** for many languages, and tools like **Neotest** (for running tests) and **Overseer.nvim** (task/job runner) [9]. Additional handy plugins include **Grapple** (for quick file bookmarking) and **Markview** (improved Markdown preview) [10] [11].
- **Platform Compatibility:** Requires Neovim 0.11+ and a **Rust toolchain** (to build the native parts like Blink.cmp) [12]. This means it works on any system where Rust and Neovim run (Linux, macOS, Windows with Rust). There is a note that using Avante's local AI model requires Docker and Ollama on your system [13] (Ollama provides local LLMs), which currently is a personal setup. For most users, internet-based Copilot will work out of the box. In practice, the config has been used on Linux and should be compatible with Mac; Windows may need WSL due to the use of some Unix-specific dependencies.
- **LazyVim Integration: Extends LazyVim natively** – abzcoding's `init.lua` imports LazyVim and then adds custom plugin specs on top of it. It explicitly credits LazyVim for "solid plugin management and defaults" [8], meaning all LazyVim keybindings and base config apply. The integration is smooth: for instance, pressing `<leader>L` opens LazyVim's menu, and you update plugins via `:Lazy`. Despite heavy additions, it remains compatible with LazyVim updates (the author regularly updates to track new LazyVim releases).
- **Unique Features: "Agentic" coding workflow** – thanks to Avante, you can use Neovim almost like an **autonomous coding agent**. Avante.nvim allows natural language commands to modify code,

generate functions, etc., within Neovim (emulating features of Cursor or Ghostwriter IDEs). This config also emphasizes performance: minimal startup time and optimized redraw (it disables expensive Vim features by default). The inclusion of advanced navigation and project tools (like **zoxide** integration for quickly jumping directories, and **Lazygit** integration for git) further streamlines automation. In summary, *abzcoding/nv* is a showcase of pushing Neovim + LazyVim to "**AI-augmented IDE**" status – blending the power of LazyVim's ecosystem with cutting-edge AI plugins [8] [10].

## gauravfs-14/NpVim – LazyVim for Web Dev with Tabnine

- **GitHub:** gauravfs-14/NpVim (☆ 4, created Oct 2025)
- **Description:** A Neovim configuration based on LazyVim tailored for **JavaScript/TypeScript web development**. NpVim stands for "Node/React tailored Vim" – it comes pre-configured for JS/TS, React, Next.js, Tailwind CSS, etc., and even includes AI code completion via Tabnine [14]. It aims to transform Neovim into a *full-fledged IDE* for front-end developers with minimal setup.
- **Notable Plugins:** All relevant LSP servers for front-end are included (TS/JS LSP, JSON/YAML, CSS, Tailwind, ESLint, etc. pre-installed). **Tree-sitter** grammars for these languages are ensured. It features **Tabnine** for AI autocompletion [15], giving VS Code-like code suggestions using machine learning. Standard dev-enhancing plugins are present: e.g. **emmet-ls** for HTML autocompletion, **eslint.nvim** and **prettier** for linting/formatting on save, a Markdown preview plugin for documentation, and **Git** integration (LazyVim's default Gitsigns and optionally Lazygit). The config inherits all LazyVim core plugins (Telescope, which-key, etc.), providing a rich base.
- **Platform Compatibility: Cross-platform** – The repo provides installation steps for Linux/macOS (shell commands) and **Windows** (PowerShell commands) separately [16] [17], indicating it's been tested on Windows 10/11 (likely via $LOCALAPPDATA\nvim). The requirements are just Neovim 0.8+ and a Nerd Font. The config lists recommended terminals on each OS (Kitty/WezTerm/Alacritty on *nix, iTerm2 on Mac, etc.) [18] for truecolor support, but it will work in any modern terminal. There's no explicit mobile support, but one could use this setup in a Codespaces or Termux environment since it's light and CLI-based.
- **LazyVim Integration: Built on LazyVim Starter** – NpVim uses LazyVim's plugin management via **lazy.nvim** and imports the core LazyVim defaults. The author notes that it's easy to extend since you can just drop in new plugins through lazy.nvim's declarative config [19]. LazyVim's updates can be pulled in by updating the lazy-lock file. Essentially, NpVim is a *specialization of LazyVim* for a web stack: it doesn't fork LazyVim, but rather layers additional language support and an AI plugin on top.
- **Unique Features: Front-end focus & AI completions** – NpVim delivers a very curated experience for web developers. For example, out of the box it has **Emmet** for HTML/JSX expansions, integrated **Tailwind CSS IntelliSense**, and project-level ESLint/Prettier handling – things you'd normally spend time configuring. The inclusion of Tabnine means you get *AI suggestions offline* (Tabnine can work locally), which is great for privacy or working in corporate settings. This config also demonstrates **Windows compatibility** with LazyVim (even adding some Windows-specific install notes), which is valuable for users who want Neovim IDE on a PC. Overall, NpVim shows how LazyVim can be adapted to a specific domain (in this case, web dev) and enhanced with AI tooling for productivity [14] [20].

## LazyIde (doctorfree/nvim-LazyIde via Lazyman)

- **GitHub:** doctorfree/nvim-LazyIde (☆ 3, Oct 2025)

- **Description: LazyIde** is an advanced IDE-like Neovim config that builds on LazyVim and is installed/configured via the *Lazyman* Neovim manager. It originated from a LazyVim starter by @jellydn and has been “extensively enhanced for use with Lazyman” [3]. The idea is to provide a full-featured, ready-to-use IDE config (for multiple languages) that can be managed alongside other configs using Lazyman’s menu system.
- **Notable Plugins:** This configuration is *loaded* with plugins across categories, essentially aiming to include everything a developer might need. For example:
- **LSP & Treesitter:** All common language servers are supported (see built-in list of ~25 servers like `pyright`, `rust_analyzer`, `tsserver`, etc.) and Treesitter grammars for dozens of languages [21] [22]. Mason.nvim is included for managing LSP/DAP tools [23].
- **Completion & Code Assist:** It integrates **zbirenbaum/copilot.lua** for GitHub Copilot AI completions [24]. Also includes nvim-cmp and many completion sources.
- **Debugging:** Has **nvim-dap** and **nvim-dap-ui** for debugging capabilities [25].
- **UI Enhancements:** navic and statuscol for winbar and number columns [26], multiple color schemes (Tokyo Night, Catppuccin, Dracula, etc.) [27], statusline (Lualine), bufferline, and Noice.nvim for improved command-line UI [28].
- **Productivity:** Telescope with Zoxide extension, Harpoon (marks) [29], project.nvim for project management [30], Spectre for search/replace [30], Yanky for yank history, Zen-mode for focus, etc. It also bundles test runners (neotest) [31], session management (persisted.nvim), and more – essentially a **kitchen-sink IDE**.
- **Platform Compatibility:** Because it’s installed via a Bash script (`lazyman.sh`), it’s primarily aimed at **Linux and macOS** users (or WSL for Windows). Lazyman automates installing dependencies and can set up multiple configs side by side. The LazyIde config itself is cross-platform (all plugins are Lua/Vimscript and platform-agnostic), though some plugin dependencies (like `ripgrep`, LSP servers) need to be available on your OS. The README provides an example of configuring the `$NVIM_APPNAME` and alias to isolate this config [32], which is useful if you want to use it on a remote server or a termux environment without clobbering an existing config.
- **LazyVim Integration: LazyVim extended via “extras”** – LazyIde uses LazyVim as the base (it even includes the entire LazyVim plugin list as an “import”). But it goes further by enabling many of LazyVim’s optional **extras modules** (for example, language extras for Go, Python, Rust, etc., UI extras, etc.). The config is structured to be managed by Lazyman, meaning you don’t manually run `:Lazy` – instead Lazyman’s script sets it all up. In essence, this is a **showcase of LazyVim’s modularity**: by turning on a lot of optional features and adding external plugins, it demonstrates how far you can go without abandoning the LazyVim framework.
- **Unique Features: Modularity and Multi-Config Support** – LazyIde’s biggest draw is that it’s part of the **Lazyman** ecosystem. You can install it alongside other configurations (AstroNvim, LunarVim, etc.) and quickly switch between them using `nvims` (fuzzy selector) [33]. This suits developers who want to experiment or use different configs for different projects. Also unique is the comprehensive nature: virtually every aspect of Neovim is configured – from Rust tools [24] to Golang support [26] [34] to writing tools. It even provides quality-of-life tweaks (e.g., Mac key-repeat settings in the README) and detailed docs via the Lazyman.dev site. If you want a *“ready-made IDE”* on top of LazyVim that you can toggle on/off, LazyIde is a strong candidate. (Do note its breadth means it’s one of the heavier configs in terms of plugin count.)

## pfassina/lazyvim-nix – Declarative LazyVim on NixOS

- **GitHub:** pfassina/lazyvim-nix (☆ 85)

- **Description:** A **Nix flake** that packages LazyVim itself as a Home Manager module, allowing declarative installation of a LazyVim-powered Neovim on NixOS. This project (started in 2025) automatically tracks LazyVim releases and pins plugin versions, so you always get a known-good LazyVim config on NixOS [35]. In simpler terms, it turns LazyVim into a Nix module you can enable, for reproducible and hassle-free Neovim setup.
- **Notable Features:** *Not a traditional plugin config,* but rather an infrastructure to use LazyVim. By enabling `programs.lazyvim.enable = true` in Home Manager, you get LazyVim's entire configuration deployed system-wide. The flake exposes options to enable LazyVim "extras" for various languages (for example, you can toggle `extras.lang.go.enable = true` to include Go support, which will auto-install Go LSP, formatter, etc. via Nix) [36] [37]. It can also automatically install external dependencies for language servers (like Node, Go, Python, etc.) via Nix, controlled by flags [38] [39]. This means your Neovim + LazyVim setup is fully reproducible from a Nix flake – no `:Lazy install` step needed, as plugins are fetched by Nix.
- **Platform Compatibility:** Aimed at **NixOS and Nix** users. It works on NixOS and also non-NixOS via Home Manager if you have Nix. For example, you could use this on a macOS with Nix installed, and it would provision LazyVim in your `$HOME`. It's not directly relevant for non-Nix environments (there you'd use LazyVim's own installer). However, the idea of declarative config is portable within Nix's domain – e.g., devshells or `nix shell` could use this module to spin up an env with Neovim+LazyVim ready.
- **LazyVim Integration: Encapsulates LazyVim** – Under the hood, this flake fetches the LazyVim GitHub release zip and uses it as the Neovim config source, applying any extra plugins you enabled in Nix config. It essentially automates what the LazyVim installer does, but via Nix. Since it tracks upstream LazyVim, updates to LazyVim (new versions) are published to this flake often (with caching to ensure stability) [35]. The result is you get LazyVim's configuration as if you installed it normally, but managed by Nix (which means no runtime downloading of plugins – they're built into the Nix store).
- **Unique Features: Reproducibility and DevOps integration** – This is the go-to solution for NixOS users who want Neovim+LazyVim. It plays nicely with NixOS modules and Home Manager, so you can put your entire Neovim config under version control in your system config. It also supports *version pinning* if needed (you can lock to a specific LazyVim release) [40]. In a broader sense, `lazyvim-nix` showcases how a complex Vim config can be treated as declarative infrastructure – something highly valued in dev environments. *Alternatives:* For those not on NixOS, a similar approach is to use **Pixi** (a Nix-based package manager by Determinate Systems) to manage Neovim and plugins. In fact, some users have started integrating Pixi in their configs – for example, one macOS dotfiles repo uses Pixi to install Python and other dev tools alongside a LazyVim config [41]. This points to a trend of combining LazyVim with Nix-style environment management for consistency.

## n-crespo/nvim – Modular Dual-Mode Config

- **GitHub:** n-crespo/nvim (☆ 7)
- **Description:** A Neovim configuration based on LazyVim that emphasizes **modularity, simplicity, and speed**. What sets it apart is that it ships in *two modes:* a **"lite"** mode for a slim experience, and a **"full"** mode that enables a wider set of plugins/language supports [42]. The user can switch between lite and full by setting an environment variable, making this config adaptable to different environments (e.g. low-powered laptop vs. powerful workstation).

- **Notable Plugins:** In full mode, it includes many common plugins (LSP servers for multiple languages, extra treesitter grammars, debuggers, etc.), whereas lite mode keeps only the essentials (likely just a couple of core languages and no heavy extras). Both modes still leverage LazyVim's defaults, so you have Telescope, Treesitter, LSP, CMP, etc. configured. This config also adds some personal favorite plugins: for instance, **snacks.picker** for enhanced Zoxide integration (quick directory jumping) [43], a custom **macro** system for automating edits, and tweaks to Markdown editing and keymaps [43]. It also integrates **Spider** (improved word motions) to enhance navigation [44]. Overall, the "full" mode would have comparable functionality to LazyVim's stock (with some extras), and "lite" mode is closer to a fast Vim-with-basics.
- **Platform Compatibility:** This config is used on both Linux and Windows (it mentions WSL-specific notes and dependencies like `wslu` and `xsel` for copy/paste on WSL [45]). It uses `bob` (Neovim version manager) for installation which works on all platforms including Windows (via cargo) [46]. Because of the lite/full toggle, one can imagine using *lite mode on a mobile or low-resource environment* (like Termux on Android or a Raspberry Pi) to keep things snappy, and full mode on a development PC. All in all, it's quite portable.
- **LazyVim Integration: Inspired by LazyVim, but highly customized** – n-crespo's setup uses LazyVim as a base and then overrides a lot. It doesn't import the entire LazyVim plugin list blindly; instead it selectively disables some defaults and adds new ones. The structure is still in LazyVim style (with `lua/config` and `lua/plugins` directories similar to LazyVim's starter). Because of this, it remains compatible with LazyVim updates, but the author has made it *their own* by borrowing ideas from various sources (the README credits that some parts are borrowed/extended). Notably, the config has a `lazyvim.json` and uses **lazy.nvim** as plugin manager, so it is definitely a LazyVim-derived workflow.
- **Unique Features: Switchable configuration profiles** – The `NVIM_FULL_CONFIG=true` toggle is a rare feature among Neovim configs [42]. This agent-like behavior (Neovim adapting to environment) is great for using one config in multiple scenarios. For example, if you SSH into a remote machine with this config, you might keep it in lite mode to avoid loading heavy plugins, whereas locally you'd use full mode. Another unique aspect is the attention to small enhancements: it comes with custom **autocommands** (e.g. pluginless auto-save, trimming whitespace on save, smarter cursorline) and custom **keymaps** that add quality-of-life improvements for editing [47]. There's even a macro system and a custom color scheme included. In summary, n-crespo's config is an exemplar of **personalizing LazyVim** – it retains LazyVim's strengths but trims or extends it thoughtfully, showing how one config can serve both minimal and maximal use cases.

## Additional Notes

- **Nushell Integration:** Many developers using LazyVim also work with modern shells like Nushell. **LazyVim provides native Nushell support** via its optional extras – enabling the Nushell extra will configure an LSP server for Nushell scripts and ensure Tree-sitter installs the `nu` parser [48] [49]. Any LazyVim-based configuration (including those above) can easily activate this, thereby supporting Nushell syntax highlighting and autocompletion in Neovim. For instance, a user could add `{"LazyVim/LazyVim", import = "lazyvim.plugins.extras.lang.nushell"}` to their plugin specs to get Nushell capabilities. This means the LazyVim ecosystem is friendly to Nushell out-of-the-box, which is great for developers who use Nushell as their main shell.

- **Pixi and Reproducible Environments:** An emerging trend is pairing LazyVim configs with tools like **Pixi** (from prefix.dev) to manage development environment dependencies. Pixi allows installing compilers, linters, etc. in a reproducible way (using Nix under the hood) without full NixOS adoption. Some Neovim users have begun using this in tandem with LazyVim – for example, one dotfiles repo uses Pixi to install Python 3.13 and other tools alongside a LazyVim config [41]. This approach, similar to pfassina's Nix flake, ensures that not just the Vim plugins but also external binaries (e.g. LSP servers, formatters) are consistent across machines. Combined with LazyVim's declarative plugin setup, this yields a highly reproducible and portable IDE. Moreover, such setups often integrate **local AI tools** (like running LLMs via Ollama or OpenCode) to complement Neovim. The previously mentioned Proteusiq dotfiles, for instance, leverages local AI model serving (Ollama) and CLI tools like `llm` in addition to Pixi [50] – showcasing how developers are creating self-contained, AI-augmented dev environments around Neovim.

**Sources:** The information above is synthesized from various GitHub repositories and documentation: Folke's dotfiles and a blog reference to his LazyVim usage [1], the README and features of *abzcoding/nv* [8] [9], *jellydn/lazy-nvim-ide* template description [6], *gauravfs-14/NpVim* README emphasizing web tech and Tabnine [14] [15], the *LazyIde* Lazyman project docs [3] [24], the *lazyvim-nix* README highlighting NixOS integration [35], and n-crespo's documentation of the lite/full config switch and custom tweaks [42] [47]. Additional context on Nushell and Pixi integration is drawn from LazyVim's official extras docs [48] and user dotfiles examples [41] [50]. Each of these repositories (dated late-2025 to early-2026) represents the state-of-the-art in using **LazyVim as a foundation** for a powerful, extensible Neovim setup.

---

[1] [2] Effective Neovim setup for web development towards 2024
https://www.devas.life/effective-neovim-setup-for-web-development-towards-2024/

[3] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] GitHub - doctorfree/nvim-LazyIde: LazyVim IDE config for Neovim
https://github.com/doctorfree/nvim-LazyIde

[4] [5] [6] [7] GitHub - jellydn/lazy-nvim-ide: My LazyVim IDE config for Neovim
https://github.com/jellydn/lazy-nvim-ide

[8] [9] [10] [11] [12] [13] GitHub - abzcoding/nv: lazyvim configurations
https://github.com/abzcoding/nv

[14] [15] [16] [17] [18] [19] [20] GitHub - gauravfs-14/NpVim: Ready-to-go neovim config build on top of lazyvim. Pre-configured for JavaScript, TypeScript, React, Tailwind. Along with code completion with tabnine.
https://github.com/gauravfs-14/NpVim

[35] [36] [37] [38] [39] [40] GitHub - pfassina/lazyvim-nix: This flake provides a bleeding edge LazyVim home-manager module, allowing you to install and configure LazyVim declaratively on NixOS. It tracks LazyVim releases automatically, and uses the latest plugin versions at the time of a LazyVim release.
https://github.com/pfassina/lazyvim-nix

[41] [50] GitHub - Proteusiq/dotfiles: Waiting for the World to Change One Byte
https://github.com/Proteusiq/dotfiles

[42] [43] [44] [45] [46] [47] GitHub - n-crespo/nvim: A neovim configuration based on LazyVim with an emphasis on modularity, simplicity and speed.
https://github.com/n-crespo/nvim

48 49 Nushell | LazyVim
http://www.lazyvim.org/extras/lang/nushell