| No | Summary | Group |
| --- | --- | --- |
| 1 | <https://github.com/Altinn/altinn-broker/pull/259>  In a pull request to the Altinn.Broker repository, the developer asked ChatGPT how to use a .json file with test data in an xUnit test. ChatGPT suggested a step-by-step approach: (i) add the JSON file to the test project and configure it to be copied to the output directory, (ii) read the file in the test using File.ReadAllText, and (iii) optionally deserialize the contents into a typed object before use.  The integrated changes followed the first two recommendations closely. The .csproj file was updated with an ItemGroup to include all .json files from the test Data folder with CopyToOutputDirectory set to PreserveNewest. In the tests, inline JSON strings were replaced with File.ReadAllText("Data/<file>.json") calls, along with a Replace("--FILEID--", fileId) step for dynamic values. New JSON fixtures were added to support existing and new test cases, including a subscription validation test.  However, ChatGPT’s deserialization example was not adopted — the tests passed raw JSON strings directly to HTTP endpoints instead of creating typed objects. The PR also contained unrelated but significant controller changes to handle Azure Event Grid events and subscription validations, which were outside the original scope of the ChatGPT conversation.  Overall, the ChatGPT guidance for managing JSON test fixtures was partially implemented, with the key file handling pattern integrated but the object deserialization omitted. | (0-25] |
| 2 | <https://github.com/Hochfrequenz/kohlrahbi/pull/158>  In PR **#158** of the kohlrahbi repo, the developer asked ChatGPT how to make a deep copy in Python. ChatGPT advised using copy.deepcopy. In the merged patch, the team imported copy and used copy.deepcopy in a new helper, \_lines\_are\_equal\_when\_ignoring\_guid, to clone two AhbLine instances, null out their guids, and compare them. That helper was then used by \_keep\_guids\_of\_unchanged\_lines\_stable to preserve GUIDs for unchanged lines when dumping FlatAHB JSON.  While ChatGPT’s suggestion (use deepcopy) was applied exactly, the majority of the patch implemented additional logic: reading an existing JSON file, matching lines via first\_true, carrying over GUIDs, and refactoring the dump flow. Only ~3 LOC of the **43 additions** directly reflect ChatGPT’s guidance (the import plus two deepcopy calls) — roughly **7%**. Hence, this case falls into **Low Patch Integration**: the AI tip provided a small but pivotal technique within a largely bespoke implementation. | (0-25] |
| 3 | <https://github.com/MartinsOnuoha/what-should-i-design/pull/8>  In PR **#8**, the developer merged a large batch of seeded UI/UX design problem statements across multiple categories into the statements.db.ts file. Prior to this, the ChatGPT conversation provided a curated list of categories, example data structure, and a detailed set of 50 JSON-formatted UI/UX problem statements with title, description, and categories fields.  In the final code, the structure, property names, formatting style, and many of the exact problem statements from ChatGPT’s output were directly inserted into the file. While some items might have been slightly adapted or reordered, the majority of the added 300 LOC (all additions) reflect ChatGPT’s generated content almost verbatim. This demonstrates that ChatGPT’s contribution was not just inspirational but directly integrated into the repository with minimal alteration. | (0-25] |
| 4 | <https://github.com/Opetushallitus/ludos/pull/102>  In PR **#102** (“OPHLUDOS-145: Käyttäjä huomaa kun istunto vanhenee”), the developer asked ChatGPT how to distinguish error cases in TypeScript and received an example that introduced **custom Error classes** (e.g., SessionExpiredError) and a fetchData wrapper that throws them based on status codes.  The merged code addresses **session expiry UX**, but does **not** adopt the custom-error-class pattern shown by ChatGPT. Instead, the implementation centers on runtime behavior and routing:   * **Session-expiry flow & tests:** Playwright specs clear cookies, assert a **302** and verify a visible **“session expired”** message with a link (data-testId="session-expired-error-message") and new-tab behavior. * **UI plumbing for reauth:** New route/component **ReauthorizeSuccessful**, Notification updated to accept a **custom link component**, and ExternalLink gains a textColor prop for consistent styling. * **Data fetching change:** Forms switch to fetchDataOrReload (replacing fetchData) to **reload/redirect on expired sessions** rather than throwing a typed error. * **Error typing touch-up:** AssignmentFormButtonRow’s submitError becomes **Error | undefined**, allowing richer error payloads, but **no custom error subclasses** are introduced.   Overall, ChatGPT’s **concept** (“differentiate session expiry from generic errors”) clearly influenced the solution, but its **specific technique** (defining and throwing custom Error subclasses) was not used. | (0-25] |
| 5 | <https://github.com/SharezoneApp/sharezone-app/pull/980>  In pull request #980 of the sharezone-app repository, the developer encountered an issue where the --whats-new argument in a GitHub Actions deployment step was passed with an equals sign (--whats-new="$SHORT\_LAST\_COMMIT\_MESSAGE"), causing multiline commit messages to be parsed as null in the Dart CLI tool. Although the environment variable echoed correctly, the CLI’s argument parser failed when handling multiline strings with =.  The developer consulted ChatGPT, which diagnosed the root cause as a shell argument parsing issue and recommended using a space-delimited syntax with quoting (--whats-new "$SHORT\_LAST\_COMMIT\_MESSAGE") to ensure the multiline string is passed as a single argument. ChatGPT also suggested adding an inline comment explaining why the equals sign was omitted and drafted a short PR description summarizing the fix.  The final merged changes closely followed ChatGPT’s advice: both affected workflow invocations were updated to the space-delimited form, the explanatory comment was added nearly verbatim, and the PR description matched ChatGPT’s draft. A minor cleanup in deploy\_ios\_command.dart removed an intermediate print and accessed the argument directly when invoking publishToAppStoreConnect. | (0-25] |
| 6 | <https://github.com/Snapchain/zkevm-bridge-ui/pull/1>  In pull request #1 of the bridge-ui repository, the developer sought to make index.html dynamically reflect values from .env files in a ViteJS application. They consulted ChatGPT, which recommended using Vite environment variables (prefixed with VITE\_) and a plugin such as vite-plugin-html to inject .env values into HTML at build time. ChatGPT also demonstrated defining placeholders in vite.config.ts and populating them with process.env values.  The final implementation followed ChatGPT’s guidance closely. A new environment variable VITE\_REPLACE\_NAME was added to .env.example, populated in deploy.sh, and validated in src/adapters/env.ts. The vite-plugin-html createHtmlPlugin was integrated into vite.config.ts to inject a <title> and <meta> description tag using process.env.VITE\_REPLACE\_NAME. Application components were refactored to consume env.replaceName in place of hardcoded "Polygon zkEVM" strings across multiple views, ensuring consistent runtime branding. | (0-25] |
| 7 | <https://github.com/StreamSG/ABCWorkshop/pull/38>  In pull request #38 of the repository, the developer aimed to refine the sidebar menu implementation in the header by enforcing a stricter data model where each menu entry could either be a horizontal rule (isHr: true) or a configurable menu item with text, routing, and optional image data—but never both. The conversation with ChatGPT centered on designing TypeScript interfaces to enforce this mutual exclusivity without resorting to a discriminant property. ChatGPT initially suggested a simple union type but acknowledged a flaw when TypeScript allowed mixed objects (e.g., { isHr: true, var1: 5 }). It then proposed using the never type to prohibit incompatible property combinations, ultimately recommending an approach where ObjectWithVariables had isHr?: never to ensure separation without needing to duplicate properties in ObjectWithHr.  In the final implementation, the developer introduced a new MenuItem interface that matched this design philosophy—isHr?: true acted as a special-case flag, with documentation stating that all other properties should be ignored when present. The Angular templates and component logic were updated to replace menuLink with menuItem, adjust structural directives (\*ngIf/\*ngFor) to use isHr, and enhance dropdown and image rendering based on the new structure.  While the final changes align conceptually with ChatGPT’s guidance—achieving the intended type constraints and improving code clarity—some implementation details (e.g., handling images via nested img objects, expanded dropdown support) go beyond the original conversation. | (0-25] |
| 8 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/228>  In **PR #228** (“Tracker functionality”), the developer implemented a comprehensive workout tracking module in a React Native (Expo) application, adding new pages, components, context hooks, and UI behaviors. The merged code introduced functionality for recording exercises, sets, reps, weights, workout completion summaries, timers, and navigation updates, with significant component-level state management using useState, useCallback, and useEffect.  The ChatGPT conversation preceding this work focused entirely on **explaining JavaScript/TypeScript closures**—their definition, mechanics, benefits, and pitfalls—illustrated with examples such as makeMultiplier and createCounter. No concrete implementation patterns or code for the tracker feature were provided; the discussion was purely conceptual and educational.  While closures are a foundational JavaScript concept and were implicitly relevant to parts of the PR (e.g., callback functions capturing variables from their enclosing scope in useCallback and useEffect hooks), there is **no evidence** in the commit history or diff that the developer directly adapted code or structure from ChatGPT’s examples. The final code instead follows common React patterns for state and event handling, without integrating any of ChatGPT’s sample logic or structure.  Of the **educational content provided**, only the general understanding of closures could have indirectly influenced coding style. Since this is an indirect conceptual carryover rather than direct code adoption, the integration is minimal. | (0-25] |
| 9 | <https://github.com/VOICEVOX/voicevox/pull/1621>  In pull request #1621 of the VOICEVOX:project-s repository (“拍子周りの修正とリファクタリング”), the developer reworked time signature handling across multiple components, shifting from position-based indexing to measureNumber-based indexing, adding validation helpers (isValidTpqn, isValidBeatType), and introducing createEmptyScore initialization. This touched components like ScoreSequencer.vue and ToolBar.vue as well as helper functions in singHelper.ts.  Before making these changes, the developer consulted ChatGPT about using **Zod brand types** in TypeScript to differentiate between numeric units (e.g., seconds, milliseconds, nanoseconds). ChatGPT suggested defining branded numeric schemas (z.number().brand<...>()) to enforce stronger type safety and prevent accidental mixing of semantically distinct values, along with examples for parsing and converting between them.  The PR did not directly integrate branded type declarations or Zod parsing methods from ChatGPT’s example. However, the conceptual advice influenced the PR by reinforcing type safety principles for musical timing data such as tpqn, beats, and beatType. The maintainer acknowledged the relevance of branded typing in the PR discussion, even linking to the ChatGPT example. Ultimately, the integration was **conceptual rather than code-level**, with the refactor focusing mainly on logic restructuring and validation enhancements rather than implementing the provided branded type patterns. | (0-25] |
| 10 | <https://github.com/VOICEVOX/voicevox_engine/pull/1122>  In pull request #1122 of the VOICEVOX/VOICEVOX repository, the developer sought ChatGPT’s advice on avoiding ShellCheck rule SC2193 false positives in GitHub Actions workflow scripts. The issue arose because actionlint’s implicit replacement of ${{ matrix.os }} caused ShellCheck to interpret comparisons like [[ ${{ matrix.os }} == windows-\* ]] as “never equal,” triggering an unnecessary lint error. ChatGPT recommended assigning the GitHub Actions expressions to shell variables (e.g., OS=${{ matrix.os }} or TARGET=${{ matrix.target }}) before performing comparisons, ensuring compatibility with both actionlint and ShellCheck. An alternative quoting approach was discussed but ultimately discarded after testing.  The merged changes closely followed ChatGPT’s suggestion, systematically replacing direct comparisons with variable assignments across multiple workflow sections, adding explanatory comments to document the workaround, and making minor lint-oriented adjustments such as quoting variables in for loops and adding # shellcheck disable=SC1091 before source commands.  This case shows how ChatGPT’s targeted advice directly resolved a tooling conflict, with developers embracing the recommended pattern almost entirely to improve maintainability and reduce the need for lint rule suppression. | (0-25] |
| 11 | <https://github.com/alcionai/corso/pull/4417>  In pull request **#4417** of the repository, the developer asked ChatGPT how to check whether a JWT is still valid in Go, noting that token creation and verification might occur in different time zones. ChatGPT recommended parsing the token, reading the exp claim, and comparing it with the current time—emphasizing that Unix timestamps make the check timezone-agnostic—and provided a short example using the (now unmaintained) dgrijalva/jwt-go library.  In the PR, the team implemented a production-ready variant: a new IsJWTExpired helper that parses claims with **golang-jwt/jwt/v5**, uses ParseUnverified with RegisteredClaims, and compares expiration via t.Before(time.Now()). They also added comprehensive unit tests, including cases with missing exp, malformed tokens, and “raw” tokens not created by the Go library, plus an unrelated URL-query helper and tests. A reviewer suggested using time.Time’s Before/After instead of Unix integer comparisons—aligning with best practices and slightly diverging from ChatGPT’s initial code—while keeping ChatGPT’s core guidance (read exp, compare to now, TZ doesn’t matter) intact.  ChatGPT contributed the key approach and control flow (parse → read exp → compare), but the final solution switched libraries, APIs, and added extensive tests. | (0-25] |
| 12 | <https://github.com/globalbibletools/gbt/pull/114>  In PR **#114**, the developer asked ChatGPT how to add tags to a Swagger/OpenAPI spec to better group routes. ChatGPT proposed adding a tags: array per route (grouping into categories like **Authentication**, **Users**, **Languages**, **Glosses**, **Verses**, etc.) and showed where to place them. The merged patch applies that guidance almost verbatim: it inserts tags: under each operation across the split path files (paths/auth.yml, users.yml, user.yml, languages.yml, language.yml, translations.yml, verse-glosses.yml, verse.yml) and aligns endpoints to sensible groupings (e.g., Authentication for auth routes; Languages for language and translation endpoints; Glosses for gloss-related endpoints; Users and Verses accordingly). The PR also makes small YAML cleanups (quote normalization, example formatting) but no structural departures from the advice. The author explicitly credits ChatGPT and notes they “just had to copy the tags into the actual files” with minor tweaks. | (0-25] |
| 13 | <https://github.com/badass-courses/course-builder/pull/83>  In PR **#83** for Course Builder, the team asked ChatGPT to help model cross-DB authorship and access control while preparing to migrate content out of Sanity. ChatGPT proposed a bridge table (content\_metadata) with a JSON resources field, and moving “types” (roles, permissions, contribution/permission types, content types) into first-class MySQL tables with junctions (e.g., user\_roles, role\_permissions, user\_permissions, contributions). The merged Drizzle schema closely mirrors that plan: it introduces contentMetadata (with resources as JSON, ownership/audit fields, indexes), user-defined roles, permissions, rolePermission, userPermission, userRole, contributor credits via contentContribution + contributionType, and publish controls via contentState/contentVisibility. The patch also adds soft-delete/active flags, slugs, timestamps, and explicit relations—extensions that refine, rather than change, ChatGPT’s structure. The author even calls out the “content\_metadata … ‘sanity bridge’” idea from the ChatGPT session. | (0-25] |
| 14 | <https://github.com/ssvlabs/ssv-web/pull/277>  In PR #277 of the SSV Web application, the developer consulted ChatGPT about implementing an ordered dictionary in TypeScript to ensure consistent value ordering when using obj.values(). ChatGPT suggested using the built-in Map object to preserve insertion order, providing an example of how to store and iterate values predictably. During the PR’s review, a contributor recommended applying an ordered dictionary approach for operatorIds handling and reusing it across multiple places. The merged changes implemented this pattern by replacing array-based payload handling with a Map-based structure, introducing a createPayload method, and ensuring consistent ordering via operatorIds.map(Number).sort(...). Alongside this, the PR also refactored gas-related constants and functions, renaming gasPrices to gasLimits and updating all references from getFixedGasPrice to getFixedGasLimit and similar helpers. The adoption of ChatGPT’s Map-based recommendation appears direct and widespread across the modified files, with the ordered dictionary concept embedded in core payload creation logic. | (0-25] |
| 15 | <https://github.com/box/developer.box.com/pull/533>  In pull request #533 of the box/developer.box repository, the developer sought to document API versioning and deprecation strategies, including examples of marking methods as deprecated in TypeScript. They consulted ChatGPT, which demonstrated the standard approach using the @deprecated JSDoc tag above a method to generate IDE and linting warnings. The example illustrated replacing an oldMethod with myNewMethod while retaining the deprecated method in the codebase.  In the final merged changes, this guidance was incorporated into the “API Versioning Strategy” documentation, including a TypeScript SDK example where deprecated methods were annotated with @deprecated followed by an end-of-life date. The integration adapted ChatGPT’s generic example to match the repository’s SDK formatting and messaging style. Only the annotation pattern and explanatory context were used, while the rest of ChatGPT’s illustrative code was replaced with a domain-specific SDK example.  This case shows how AI-provided generic coding patterns can be selectively integrated into technical documentation, with modifications made to reflect organizational standards and project-specific code conventions. | (0-25] |
| 16 | <https://github.com/citusdata/activerecord-multi-tenant/pull/199>  In pull request #199 of the citusdata/activerecord-multi-tenant repository, the developer encountered an issue where sensitive parameter filtering (e.g., :password) was not working when using the activerecord-multi-tenant library in a Rails project. They consulted ChatGPT, which suggested an RSpec unit test that mocked controller parameters, invoked an action that logs them, and verified that sensitive values were excluded from logs.  In the final implementation, instead of adopting the full controller-mocking approach, the developers integrated a model-level test that confirmed ActiveRecord’s inspect method masked sensitive attributes ([FILTERED]). This was achieved by setting config.filter\_parameters in a test Rails application and ensuring it was applied via ActiveSupport.on\_load(:active\_record). The patch also restructured the code to prepend the filtering logic at the correct load point and added the password column to the test schema for verification.  Only ~20% were integrated, reflecting selective adoption of the core verification concept while replacing the broader controller/log inspection strategy with a simpler and more targeted model inspection test aligned with the repository’s testing style. This demonstrates how AI-provided test ideas can be adapted and streamlined to fit project-specific testing frameworks and scope. | (0-25] |
| 17 | <https://github.com/pyspark-ai/pyspark-ai/pull/1>  In pull request #1 of the pyspark-ai/spark-llm repository, the developer added an llm\_verify function to validate that a given PySpark DataFrame meets specified quality criteria. The developer consulted ChatGPT to understand how to capture results from Python’s exec() function. ChatGPT explained that exec() does not directly return values but can store them in a provided locals dictionary, allowing retrieval of computed results after execution.  In the final implementation, this approach was integrated almost exactly as suggested: the generated verification code was executed using exec(llm\_output, {"df": df}, locals\_), and the resulting boolean value was retrieved via locals\_['result'] for logging. This formed the core mechanism enabling dynamically generated tests to return pass/fail outcomes. | (0-25] |
| 18 | <https://github.com/dust-tt/dust/pull/508>  In the “Github connector (initial full sync, issues only)” pull request #508, the developer asked ChatGPT how to enforce exhaustiveness in a TypeScript switch by using never. ChatGPT recommended the idiomatic helper pattern assertNever(value: never): never and returning assertNever(provider) in the default branch to trigger compile-time checking and fail fast at runtime.  Reviewing the PR changes, the feature work centers on adding a GitHub connector, webhooks plumbing, API routes, and Temporal activities. There’s no evidence of an assertNever helper being introduced or the switch-based exhaustiveness refactor being applied in this patch; the only nearby TypeScript ergonomics shown are unrelated (e.g., a // @ts-expect-error in a migration). In other words, the guidance informed the discussion but didn’t translate into code in this PR. | (0-25] |
| 19 | <https://github.com/exchange-wata/next.js14-tailwindcss-udemy-blog/pull/6>  In pull request #6 of the main repository, the developer asked ChatGPT why the error object in a TypeScript form state type should be optional. ChatGPT explained that optional properties are useful in form validation because not all fields will have errors at the same time, allowing flexibility in representing per-field validation states. It recommended keeping both the container error object and its title and content arrays optional.  In the merged code, the developer refactored the type by renaming errors? to error and making the container object required, but retained optionality for the title and content arrays. The UI logic was updated to remove null checks for the container while still handling the possibility of missing field-level errors.  This reflects a partial alignment with ChatGPT’s advice—only the per-field optionality was preserved—while the key suggestion of making the entire error object optional was not implemented. Overall, the adoption rate is estimated at around 20–25%, indicating low patch integration where only a subset of the recommended design was incorporated. | (0-25] |
| 20 | <https://github.com/firezone/firezone/pull/3621>  In pull request #3621 of the firezone repository, the developer asked ChatGPT why a sign-in notification in an Android foreground service was not dismissed after tapping, despite using .setAutoCancel(true). ChatGPT explained that .setAutoCancel(true) does not work with foreground service notifications and suggested restructuring the tap handling to stop the foreground service or transition the notification out of foreground mode when the user interacts with it. It proposed using a PendingIntent to start an Activity or BroadcastReceiver that would stop the service, allowing the notification to disappear.  In the final implementation, the notification system was refactored into a dedicated TunnelStatusNotification object. This refactor moved notification configuration out of TunnelService, introduced distinct status types (SignedOut, Connecting, Connected), and ensured that SignedOut notifications used .setAutoCancel(true) with an explicit PendingIntent that launches the main activity—matching ChatGPT’s core recommendation to separate notification behavior and make tap interactions dismissible. ChatGPT’s functional guidance was incorporated, with additional structural improvements and type-safe status handling added by the developer. | (0-25] |
| 21 | <https://github.com/apache/kafka/pull/14216>  In pull request #119 of the admin-league module, the developer sought ChatGPT’s help to resolve a bug where clicking the “삭제” (delete) button in a list of leagues would always delete the last league in the loop rather than the intended one. ChatGPT explained that the issue stemmed from how closures in JavaScript capture loop variables and recommended storing the selected league’s ID in state (selectedLeagueId) to ensure the modal and delete action target the correct item. It also proposed alternative fixes using event.stopPropagation() and generated delete handlers to capture the correct ID without extra state.  In the final merged implementation, the developer adopted ChatGPT’s primary recommendation—introducing a selectedLeagueId state, modifying toggleModal to accept leagueId as a parameter, and updating modal rendering to only display when selectedLeagueId matches the current league. The delete button inside the modal now calls deleteLeague with the correct ID, ensuring the intended record is deleted. | (0-25] |
| 22 | <https://github.com/jaoafa/VCSpeaker.kt/pull/70>  In PR #70 of the vcspeaker-kt project, the developer asked ChatGPT for Kotlin code using **sksamuel/scrimage** to draw white text with a black outline on an image. ChatGPT returned an example that mixed Scrimage with raw AWT drawing and achieved the “outline” by drawing the text at offset positions around the main glyph.  In the merged changes, the team **did adopt Scrimage**, but implemented a different pipeline tailored to their feature: OCR via Google Vision and visualizing results. They:   * Added the Scrimage dependency and used ImmutableImage/PngWriter. * Built a VisionApi.drawTextAnnotations method that overlays **semi-transparent rectangles** (FilledRect) and draws **plain white text** (Text) at detected bounding boxes. * Replied in Discord with the annotated image; no outlined text was used.   Only the high-level idea (“use Scrimage to draw text on images and save the result”) overlapped; core details from ChatGPT (AWT graphics, multi-offset outline trick, the exact draw flow) were **not** integrated. Roughly ~15–20% of the suggested approach influenced the final patch (dependency choice, image creation/saving, drawing text), while the specific outlined-text technique and AWT usage were replaced by Scrimage’s canvas drawables and project-specific annotation logic. | (0-25] |
| 23 | <https://github.com/jsdom/jsdom/pull/3644>  In jsdom PR **#3644** (“Proposal: New CSS selector engine for jsdom”), the developer asked ChatGPT to **rewrite a Mocha/Jsdom test into Web Platform Tests (WPT) format** for issue **#1163** (matching div[title=""]). ChatGPT produced a WPT-style HTML file using testharness.js and assert\_array\_equals, but it **split the assertions into two test() blocks** instead of following the user’s stated preference to **combine all original it() blocks into one test()**. In the merged PR, the core change was a **selector engine swap** (from nwsapi to @asamuzakjp/dom-selector) plus benchmarks and helper rewrites; there’s **no evidence that ChatGPT’s converted WPT test was added**. At best, the response offered formatting guidance for WPT tests, but **no tangible lines from it landed in the patch**. | (0-25] |
| 24 | <https://github.com/viets-software-club/truffle-ai-backend/pull/52>  In PR **#52** (“[BACK-69] Improving Categories and making them more specific”), the developer consulted ChatGPT for a **TypeScript enum-based representation of software project categories** and examples of variables typed as arrays of that enum. ChatGPT returned an enum ProjectCategory with five string-valued categories (Web, Mobile, AI, Data, Security) and demonstrated array-typed variables holding these values. In the actual code changes, however, the team implemented a **numeric enum** (Topic) with category IDs mapped to names (MachineLearning = 1, DevTools = 2, Infrastructure = 3, Miscellaneous = 8, CategorizationError = 9) and designed their categorization pipeline to process numeric responses from GPT, avoiding string-based labels to prevent hallucinated category names. While the general concept of “an enum for project categories” aligns with ChatGPT’s suggestion, the implementation diverged substantially in naming, values, and integration with the broader categorization logic. No direct lines from ChatGPT’s code were adopted, making the influence **conceptual only**, with an estimated integration rate of **~5–10%**. | (0-25] |
| 25 | <https://github.com/langroid/langroid/pull/301>  In PR **#301** (“support neo4j to make query and write to a DB”), the developer consulted ChatGPT for Python scripts that (i) retrieved a GitHub repository’s latest version using **PyGithub**, and (ii) fetched the last 10 version numbers of a Python package from **PyPI** via its JSON API. ChatGPT’s examples included authentication with GitHub’s API, reading tags to determine the latest version, and querying PyPI’s release data with requests for version sorting and selection.  In the merged changes, the developer incorporated this logic into a **DependencyGraphAgent**-based system for building and querying package dependency graphs in **Neo4j**. The production code included DepGraphTool and GoogleSearchTool integrations that replicated the ChatGPT-suggested flow for fetching package versions—both from GitHub and PyPI—before constructing the dependency graph. While the actual code was refactored for the Langroid framework (e.g., tool messages, task orchestration, and environment-variable-based config), the **core approach to version retrieval from ChatGPT’s examples was retained almost verbatim**.  This high alignment indicates the AI-generated logic was directly transferable, with modifications mainly for framework compatibility rather than algorithmic changes. | (0-25] |
| 26 | <https://github.com/pass-culture/pass-culture-main/pull/9227>  In PR **#9227** (“handle duplication during offer creation/edition”), the developer asked ChatGPT how to remove an **N+1 query** in stock upserts. ChatGPT proposed: (1) prefetch all potentially matching rows in **one query** by OR-ing together ANDed field tuples, and (2) check existence **in-memory** during the loop. The merged patch implements this almost verbatim: it introduces \_get\_existing\_stocks\_by\_fields(...) building a list of and\_(...) predicates and combining them with or\_(...), returns all matches in a single call, and then uses a pure-Python \_stock\_exists(...) to gate edits/creates. The PR also adds a **feature toggle** path, adjusts response typing, and expands tests to assert “no duplicate create/edit” behavior—engineering hardening around the same core idea. Overall, the final solution mirrors ChatGPT’s approach closely, with project-specific refinements (time zone normalization, price defaults, pagination FF). Integration is high (most of the suggested structure and logic were adopted). | (0-25] |
| 27 | <https://github.com/pokt-network/poktroll/pull/185>  In pull request #185 of the pokt-network/poktroll repository, the developer asked ChatGPT if Go had a built-in function to remove leading spaces from multiline strings while keeping the source code readable. ChatGPT suggested an approach using strings.Split, strings.TrimLeft, and strings.Join to strip indentation, then encapsulated the logic in a helper function trimString(...) for cleaner usage. ChatGPT provided a complete runnable example showing how to integrate the helper in a Go program.  However, the final PR focused entirely on adding a staking flag, config file parser, and associated CLI/test changes for supplier staking, without adding the proposed helper function or any string manipulation logic. This indicates that while the conversation explored a useful utility, it was ultimately unrelated to the developer’s actual code changes in the merged PR. | (0-25] |
| 28 | <https://github.com/pyspark-ai/pyspark-ai/pull/1>  In pull request #1 of the gengliangwang/spark-llm repository, the developer asked ChatGPT how to retrieve the result of Python’s exec() function. ChatGPT explained that exec() itself returns None but that results can be captured by passing a locals dictionary and then retrieving variables from it after execution. The explanation included a working example that assigned a value to result inside executed code and then accessed it via locals\_['result'].  In the PR, this approach was directly applied in the newly introduced verify\_df method for the SparkLLMAssistant class. After generating PySpark verification code from an LLM, the method executed it with exec(llm\_output, {"df": df}, locals\_) and retrieved the verification outcome from locals\_['result']. This matched ChatGPT’s suggested implementation pattern exactly, with only contextual adjustments for DataFrame verification. | (0-25] |
| 29 | <https://github.com/pyspark-ai/pyspark-ai/pull/157>  In pull request **#157** of the pyspark-ai/pyspark-ai repository, the developer asked ChatGPT for a Python class that manages files under a directory with an **LRU eviction policy** when a **max total size** is exceeded. ChatGPT proposed an LRUFileCache using collections.OrderedDict to (i) track filenames and sizes, (ii) move accessed entries to the end, and (iii) evict with popitem(last=False) until under the limit, updating a running byte count.  The merged PR introduced an LRUVectorStore that closely mirrors this design and integrates it into the vector search toolchain:   * **Core design adopted**: OrderedDict-backed LRU, current\_size tracking, access() to move\_to\_end, and add() with an **eviction loop** using popitem(last=False); deletions via shutil.rmtree. * **Initialization** scans an existing directory and seeds the cache (a refinement over the example). * **Byte-accurate limit**: the class converts a GB setting to bytes (max\_bytes) and compares against cumulative size. * **Integration**: Wired into SimilarValueTool/VectorSearchUtil and constructed from SparkAI via new params (vector\_store\_dir, vector\_store\_max\_gb). * **Tests**: Added comprehensive unit tests covering eviction order, pre-populated directories, zero-capacity behavior, and storage accounting (with get\_file\_size\_bytes mocking).   While naming, units (GB→bytes), and directory handling were adapted to project needs, the **algorithmic core** from ChatGPT’s answer was preserved almost verbatim. | (0-25] |
| 30 | <https://github.com/qin-team-recipe/05-recipe-app/pull/62>  In pull request #62 of the project’s front-end module, the developer asked ChatGPT whether the next/link component should be used for external URLs in a Next.js 13.4 application using the App Router. ChatGPT explained that next/link is intended for internal navigation (to enable client-side transitions and prefetching) and recommended using a regular <a> element for external URLs, optionally with a custom component that switches between <Link> and <a> depending on the URL.  The developer’s final implementation of LinkCard.tsx closely mirrored ChatGPT’s advice, rendering all external URLs with <a href={url}> and not using next/link for them. Additional UI work included social-media-specific icons via react-icons, Tailwind gradient styles for Instagram, and integrating the component into the /chef/[chefId]/link page.  Integrated with minor adjustments to match the project’s design system and omit outdated passHref usage. This reflects alignment between ChatGPT’s recommendations and the final merged changes, with the remaining differences attributable to framework version best practices and UI refinements. | (0-25] |
| 31 | <https://github.com/seamapi/seam-cli/pull/68>  In PR #68, the team needed interactive editing for array-type parameters (e.g., access\_code\_ids) in a CLI flow. ChatGPT supplied a minimal Node.js example using the prompts library to maintain a list with **add/remove/exit** actions, showing the live list and mutating it via push/splice. The merged patch lifted this approach almost wholesale but productionized it:   * Extracted the loop into a reusable TypeScript helper interactForArray(array, message): Promise<string[]> with a clear **Add / Remove / Done** menu, live list preview, and **index validation**. * Integrated it into the existing OpenAPI prompt flow by switching array handling in interact-for-open-api-object.ts from one-off text input to the new array editor for any type: "array" schema. * Kept the same core mechanics (loop + select menu + push/splice) while adding small UX improvements (1-based indices, empty-state messaging).   ChatGPT’s pattern was adopted; the project added typing, reuse, and validation. | (0-25] |
| 32 | <https://github.com/timjuic/paypro-android/pull/8>  In PR #8 of the Found404/paypro repository, the goal was to implement merchant deletion via a modal dialog rather than as a standalone page. The developer consulted ChatGPT for guidance on converting an existing DeletingMerchants page into a Jetpack Compose Dialog. ChatGPT proposed a Kotlin implementation using Dialog with a text field for the merchant name, confirmation/cancellation buttons, and keyboard handling, closely mirroring the existing layout but within a modal context.  In the merged changes, the team introduced a new DeleteMerchantPopup composable that encapsulates the modal’s UI and behavior. This included:   * A TextField for the merchant name, validated against the actual merchant’s name before deletion. * Integration with MerchantService.deleteMerchant() to perform backend deletion. * Success/failure feedback via Toast. * Confirm and Cancel buttons invoking callbacks to update UI state. * Embedding this popup into MerchantItem, triggered by an edit icon, while keeping merchant list interactions intact.   While the production version added API calls, input validation, and toast notifications, the core modal concept, layout structure, and trigger mechanism came directly from ChatGPT’s suggestion. | (0-25] |
| 33 | <https://github.com/uchicago-cs/chigame/pull/265>  In PR #265 of the chigame/chigame repository, the developer asked ChatGPT how to “add a token to the user every month” in Django, requesting an implementation using Django models/views. ChatGPT proposed a full solution: a separate UserProfile model with token\_count and last\_token\_addition, a View (AddTokenMonthlyView) that iterates users and updates tokens when a month boundary passes, a URL route, and an external scheduler (cron/Celery) to trigger it.  What actually landed was far lighter-weight and project-specific: no new model or URL endpoint, and no calendar logic. Instead, the PR added a simple distribute\_tokens() function in games/views.py that filters users with tokens < 3, increments tokens by 1, and saves—intended to be run manually once a month (per reviewer guidance). Comments even removed the placeholder date check and linked the ChatGPT conversation as reference, emphasizing a manual monthly run rather than automated scheduling. The rest of the PR focused on tournament UX changes (creator tracking, permissions, list/detail gating), unrelated to token automation.  Overall, only the **broad idea** (“use Django code to add tokens”) was adopted; the **structural pieces** ChatGPT suggested (new model, month-diff logic, dedicated view + URL, scheduled trigger) were not. Roughly ~8–12 lines of the AI-inspired logic were integrated versus ~40–60 lines proposed, for an estimated **~15–20% integration**. This reflects a deliberate choice to fit existing domain fields (User.tokens) and a manual ops cadence over introducing new persistence and scheduling complexity. | (0-25] |
| 34 | <https://github.com/viets-software-club/truffle-ai-backend/pull/52>  In PR #52 of the graphql\_server package, the developer asked ChatGPT for a simple TypeScript example: define an enum with five categories for software projects, then initialize array variables of that enum type. ChatGPT delivered exactly that — a ProjectCategory string enum with Web, Mobile, AI, Data, and Security, plus three array-typed variables (project1, project2, project3) initialized with combinations of those values.  What actually landed in the PR was a much more complex refactor of project categorization logic in openAIApi.ts, not a direct adoption of the example. The merged code did introduce an enum, Topic, but:   * It used numeric values (1, 2, 3, 8, 9) mapped to specific topic labels like MachineLearning, DevTools, Infrastructure, etc., rather than ChatGPT’s five high-level categories. * There were no direct enum-array variable declarations like in ChatGPT’s snippet; instead, the enum was consumed dynamically in helper functions (getCategoryFromGPT, createCategorizationPrompt) and mapped from GPT responses. * The PR’s broader changes removed the original categorizeProjectGeneral/categorizeProjectSpecific functions and replaced them with a streamlined numeric category mapping system, integrating with GitHub topics and descriptions.   Thus, while the *concept* of “using a TypeScript enum to represent project categories” clearly survived from ChatGPT’s answer, the actual values, style, and usage were completely reworked to fit the project’s AI-driven categorization workflow. The enum’s role became part of an automated classification pipeline rather than a static typed list for manual assignment. | (0-25] |
| 35 | <https://github.com/BitBoxSwiss/bitbox-wallet-app/pull/2368>  In pull request #2368 of the digitalbitbox/bitbox-wallet-app repository, the developer initially consulted ChatGPT about a “hacky” use of a MutableLiveData in an Android ViewModel to trigger authentication requests. ChatGPT recommended replacing the boolean flag with a more explicit, event-based approach, such as an event wrapper or a dedicated AuthRequestEvent class, to avoid re-triggering after configuration changes and to make the intent clearer.  While the actual patch went well beyond the small Android snippet, it followed the same architectural principle—removing the boolean hack and implementing a robust, event-driven authentication mechanism. The final changes introduced a unified backend–frontend authentication flow: new Go backend types and handlers for auth-required, auth-ok, and related events; new API endpoints for triggering, forcing, and reporting authentication results; platform-specific environment Auth() methods; Android biometric authentication integration with BiometricAuthHelper; LiveData-based UI handling in MainActivity; and a React web UI component for handling auth events.  The core idea from ChatGPT—using explicit events for auth requests instead of boolean state—was fully adopted in a cross-platform implementation. | (25-50] |
| 36 | <https://github.com/Mudlet/Mudlet/pull/7123>  In pull request #7123 of the Mudlet/Mudlet repository, the developer fixed a crash that occurred when double-clicking on a word in TTextEdit. The bug was caused by an out-of-bounds access to QString::at() inside expandSelectionToWords() when xind or yind were invalid.  After reviewing the stack trace, the developer consulted ChatGPT, which identified the root cause and suggested adding explicit index bounds checks. ChatGPT’s proposed solution included (i) validating yind against the size of lineBuffer, (ii) ensuring xind stayed within the current line’s length in both loops, and (iii) breaking early when indices were invalid.  The merged patch (+25/−8 LOC) applied the same defensive programming strategy—wrapping both loops with yind range checks, validating xind before accessing characters, and using a local const QChar for clarity. The main differences were minor stylistic adjustments to match project conventions, with no broader refactoring beyond the targeted fix.  Of the 39 LOC suggested by ChatGPT, 18 LOC (46.2%) were integrated. This reflects substantial reuse of the AI-provided logic while adapting it to the project’s coding style, placing the case in the lower-median integration range. | (25-50] |
| 37 | <https://github.com/QRTaxi/QRTaxi_Web/pull/29>  In pull request #29 of the relevant repository, the developer refactored WebSocket handling and API call logic, focusing on fixing an issue where a visibilitychange event listener was not being removed as expected when the WebSocket closed cleanly. The root cause was that the listener was registered with an anonymous function, making it impossible for removeEventListener to match and remove it later.  The developer consulted ChatGPT, which explained why anonymous functions prevent proper removal and suggested defining the listener as a named or stored function reference. The guidance included creating a reusable listener function that captures necessary parameters while avoiding redundant wrapper nesting, thereby ensuring that the same function reference could be passed to both addEventListener and removeEventListener.  The final merged patch implemented this strategy by assigning the listener to a const variable, adding it after socket.onopen, and removing it in the WebSocket onclose handler. The rest of the changes involved broader refactoring—simplifying Axios error handling, restructuring type definitions, and cleaning up conditional routing logic. This reflects strong reuse of the AI-provided solution, adapted within a larger code cleanup and refactoring effort, placing it in the upper-median integration range. | (25-50] |
| 38 | <https://github.com/StuartGavidia/businessapp/pull/35>  In pull request #35 of the ​​StuartGavidia/businessapp repository, the developer introduced a global loading indicator to display a spinner during asynchronous operations across multiple pages in a React application. The goal was to enhance the user experience by visually signaling that data was being fetched or processed before rendering content.  The developer consulted ChatGPT, which recommended implementing a global loading state using React Context. The proposed approach included a LoadingProvider component, a useLoading hook for state access, and a reusable Spinner component with CSS animations. This setup would allow centralized control over the loading state, enabling consistent spinner behavior throughout the app.  In the final merged PR, the developer adopted the reusable spinner concept almost exactly as suggested—creating a Spinner component (Spinner.tsx + Spinner.css) styled with animated borders, and integrating it into App.tsx. However, instead of using React Context as proposed, they opted for a simpler implementation by managing the loading state locally within App.tsx, with a simulated delay to demonstrate spinner behavior. This choice reduced complexity but limited the spinner’s true global applicability without further refactoring.  The adopted code covers the visual and structural aspects of ChatGPT’s solution while omitting the context-based global state management. | (25-50] |
| 39 | <https://github.com/UNLV-CS472-672/2024-S-GROUP1-Roadwatch/pull/107>  In pull request #107 - UNLV-CS472-672/2024-S-GROUP1-Roadwatch repo, the developer requested ChatGPT’s help in creating a reusable React “post card” that displayed a title, a truncated body preview, and a “View Post” button, preferably using Material UI (MUI) components. ChatGPT responded with a compact PostCard component built with Card, CardContent, Typography, and Button from @mui/material. The implementation accepted { title, body, onViewPost } as props, used a 100-character substring for body previews, and called onViewPost when the button was clicked.  In the merged code, the team introduced a CommunityPost component (client/src/components/CommunityPost/CommunityPost.tsx) along with a scoped SCSS module and an index export. While the final component preserved ChatGPT’s structural ideas—MUI Card layout, title and body preview, and a “View Post” trigger—it expanded the design considerably. The adopted version added:   * Marker-based post support with selectable icons for different types (e.g., Cone, CarAccident, Pothole). * An avatar for the post creator. * Body previews styled with CSS max-height and overflow rather than string truncation. * A CustomButton component instead of MUI’s default Button. * A broader prop model (isMarker, content, user, marker?) and richer styling in CommunityPost.module.scss.   Of ChatGPT’s suggested 22–24 LOC, about 9–11 LOC (roughly 40–45%) were directly integrated, with the rest replaced or restructured to match the project’s UX and design system. This places the integration in the **lower-median** range, as ChatGPT provided a useful skeleton that the developers heavily adapted to meet additional feature and styling requirements. | (25-50] |
| 40 | <https://github.com/UNLV-CS472-672/2024-S-GROUP1-Roadwatch/pull/120>  In pull request #120 of the UNLV-CS472-672 repository, the developer sought ChatGPT’s help to remove redundancy from the CreatePost component, which originally contained six nearly identical Material UI FABs for selecting markers. ChatGPT proposed replacing the repeated JSX with a loop over a markers array, dynamically rendering each FAB, applying an active style when the selected marker matched, and using a shared click handler.  The final merged code in CreatePost.tsx closely follows this advice. It replaces the hardcoded FAB blocks with a .map() over an array of marker IDs (['cone','pothole','roadDamage','carAccident','warningSign']), conditionally applies the active class, and passes a single markerClick handler to update state. The main differences are that the developer used an inline array instead of an array of objects, selected icons via a conditional src mapping instead of storing them directly in the array, and integrated the logic into a larger feature-rich component that included a swipeable drawer, form inputs, and geolocation support. This reflects that the AI’s core structural recommendation was implemented with only minor adaptation for asset handling and broader UI context. | (25-50] |
| 41 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/208>  In pull request #208 of the main branch, the developer sought ChatGPT’s help to address two specific UI challenges in the MessageView component:  (1) Aligning chat bubbles left or right depending on whether the sender ID matched the logged-in user ID.  (2) Dynamically resizing the chat bubble width based on the amount of text it contained.  ChatGPT proposed using strict equality (===), conditional alignSelf styles for left/right alignment, and a calculated maxWidth tied to message length to achieve dynamic resizing. The final merged implementation adopted the alignment logic by conditionally applying self-end for the current user’s messages and self-start for others. However, instead of calculating bubble width from text length, the developer opted for a fixed w-1/2 Tailwind width class, resulting in consistent bubble sizes regardless of message length.  Of the ~24 LOC suggested by ChatGPT for these features, about 9 LOC (37.5%) were integrated. The partial adoption indicates that while ChatGPT’s solution guided the alignment fix, the developer chose a simpler, static width approach for sizing, possibly for design consistency or ease of implementation. | (25-50] |
| 42 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/223>  In pull request #223 of the main branch, the developer aimed to implement selection-based filtering for exercises in the “create new workout” and “start existing workout” pages. The ChatGPT discussion began with rendering an array of equipment names as TouchableOpacity elements, progressed to explaining the use of the index parameter in .map(), addressed a JSX error caused by lowercase component naming, and culminated in logic for returning either the full list or a subset of selected items. ChatGPT proposed using a state map for selections, a selectToggle(eid) helper, a derived filteredList computed from the selection state, and an optional useEffect to keep this list updated.  The merged code in exerciseList.tsx adopted the core pattern of mapping items to TouchableOpacity with background color determined by a selection state map, stable keys (IDs instead of indices), and the presence of a filteredList. However, instead of using the “selected only vs. full list” toggle, the implemented filteredList was tied to a SearchBar component for text-based filtering, and selection state was used only for visual highlighting. The selection-driven filtering described by ChatGPT was not yet wired into the returned list, and was noted as a future task in the PR description.  The AI’s guidance provided the structural foundation for rendering, selection handling, and list derivation, while the developer aligned filtering with existing search-driven UX and deferred integrating selection-based filtering. | (25-50] |
| 43 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/258>  In pull request #258, the team built a “Select Exercises (by muscle group)” flow and related UI wiring. The ChatGPT exchange covered (a) how JSX && conditional rendering works, (b) what “truthy” means in JS, and (c) simple state patterns for selectable lists—specifically a boolean map keyed by ID with selectToggle(id) and a reset via setSelect({}). The merged code created a new /filteredExercises screen that filters by bodyPart, added a right-side “Add(n)” action in the nav bar, and implemented selection with highlight styles. However, the final selection model used an array (selectedExercises.includes(id)) instead of the boolean map pattern, and only a small slice of ChatGPT’s suggestions (conditional render, toggle, and reset approach) directly influenced the patch. Roughly 8 of about 20 suggested lines (≈40%) were effectively adopted, placing this in the lower-median integration range: ChatGPT’s guidance informed the selection/conditional basics, while the developers chose different data structures and broader UI changes to fit their codebase. | (25-50] |
| 44 | <https://github.com/VOICEVOX/voicevox/pull/1836>  In pull request #1836 of the VOICEVOX repository, the developer asked ChatGPT to refactor a TypeScript block that determined the latest command timestamp from two undo stacks, aiming for simpler and more readable code. ChatGPT recommended introducing a helper function to extract the last element’s unixMillisec from any given stack, then computing the maximum of the two timestamps when both were present, otherwise returning whichever was non-null, using concise conditional logic.  The merged code in src/store/command.ts closely mirrored this suggestion: it added a getLastCommandUnixMillisec helper, used it to retrieve times for “talk” and “song” editors, and returned the latest timestamp via Math.max(...) with explicit != null checks to avoid truthiness pitfalls.  This adoption demonstrates how the AI-assisted refactor fit seamlessly into the new multi-editor undo/redo architecture without requiring major modifications. | (25-50] |
| 45 | <https://github.com/annuaire-entreprises-data-gouv-fr/site/pull/796>  In pull request **#796** of the target repository, the developer sought clarification on checking for null or empty values in TypeScript. In a ChatGPT conversation, they were reminded that an empty array ([]) is truthy in JavaScript/TypeScript, so a simple !array check only detects null or undefined but **not** an empty array. ChatGPT recommended using !array || array.length === 0 when explicitly testing for both null and emptiness.  In the merged changes, conditional logic across the updated code reflects this best practice, ensuring that list-like data structures are only considered valid if they are both non-null and contain elements. The revision avoids the earlier misconception that !array alone would detect emptiness. | (25-50] |
| 46 | <https://github.com/dotCMS/core/pull/25432>  In pull request **#25432** of the dotCMS/core repository, a reviewer flagged that using a <li> element directly as a clickable control in an Angular component introduced accessibility issues, since list items are not inherently interactive and lack keyboard operability. The reviewer linked to a ChatGPT discussion where it was recommended to replace clickable <li> elements with semantically appropriate interactive elements, such as <button> or <a> tags, to ensure proper keyboard focus, screen reader compatibility, and built-in accessibility features.  The ChatGPT guidance emphasized:  (i) replacing the clickable <li> with a <button> inside the list item,  (ii) preserving semantic HTML roles without requiring extra ARIA attributes or tabindex hacks, and  (iii) maintaining styling through CSS while ensuring native keyboard and assistive technology support.  In the merged PR, the dot-device-selector-seo.component.html implementation fully adopted these recommendations — clickable list items were restructured so each <li> contained a <button> element for user interaction, rather than binding (click) directly to the <li>. This provided both semantic correctness and accessibility compliance without resorting to manual role or keyboard event handling. | (25-50] |
| 47 | <https://github.com/equinix/metal-cli/pull/405>  In pull request **#405** of the Equinix CLI repo, the author asked ChatGPT how to correctly close an os.Pipe() in Go tests and whether t.Fatal made more sense than t.Error + return, and then whether to defer the close. ChatGPT recommended (i) closing/cleaning up via defer (and restoring os.Stdout), (ii) using t.Fatal to stop the test immediately on failure, and (iii) improving error handling for io.ReadAll.  In the merged patch, the tests were updated to:   * add a **cleanup block** using t.Cleanup to **close the writer and restore os.Stdout** (functionally similar to ChatGPT’s defer advice), * switch to **t.Fatal(err)** after command execution errors (they kept a preceding t.Error(err) line, which is redundant but harmless), * leave **io.ReadAll errors ignored** (out, \_ := io.ReadAll(r)), so the suggested read-error handling was not adopted.   Net effect: the PR integrated the **resource-cleanup pattern** and the **fatal-on-error behavior** that ChatGPT advised, but **didn’t** incorporate the proposed read-error handling or buffered reads. Given that only part of the recommendations were applied—and with slight deviations—the patch reflects a **lower-median integration (~35–45%)**: ChatGPT’s guidance clearly shaped the cleanup and failure semantics, while other suggestions were skipped. | (25-50] |
| 48 | <https://github.com/annuaire-entreprises-data-gouv-fr/site/pull/796>  In pull request #796 of the repository, the developer asked ChatGPT how to check if a value was null or empty in TypeScript, specifically whether !array would detect empty arrays. ChatGPT explained various null/empty checks for strings, arrays, objects, and general variables, noting that !array only detects falsy values (such as null or undefined) and not empty arrays, which require an additional .length === 0 check.  In the final merged code, the developer applied a check in data-section-loader.tsx that validated whether dataSources was null or empty using:  !dataSources || dataSources.length === 0  While this aligned with ChatGPT’s conceptual advice, the implementation was independently written and did not copy any code from the AI’s examples. The integration rate is therefore low (~27%), as ChatGPT’s contribution was limited to confirming the correct TypeScript behavior and influencing the developer’s reasoning rather than supplying directly merged code. | (25-50] |
| 49 | <https://github.com/meta-llama/llama-cookbook/pull/279>  In pull request **#279** of the meta-llama/llama-recipes repository, the contributor asked ChatGPT for a **Python script** that replicates a curl POST to a vLLM endpoint (http://localhost:5000/generate) with a JSON payload. ChatGPT returned a short requests-based snippet that posts the JSON and prints the response, plus an installation hint for requests.  The merged PR, however, is a **documentation-focused tutorial** on on-prem inference with vLLM and Hugging Face TGI. It adds README updates and a new guide (demo\_apps/llama-on-prem.md) that demonstrates:   * standing up vLLM (both general API and OpenAI-compatible servers), * invoking the service via **curl** (including the exact /generate example), and * client integrations using **LangChain** (OpenAI-compatible path) and **Hugging Face Text Generation Inference**.   No direct Python requests.post(...) example from ChatGPT was added to the repo; the PR sticks to curl and framework integrations instead. Thus, **0 lines** of ChatGPT’s code were adopted out of roughly **12–15 suggested LOC**—an effective **0% integration**. Conceptually, the PR aligns with the same endpoint and payload pattern that ChatGPT used, but the assistant’s concrete Python snippet was not incorporated, placing this case firmly in the **Low Patch Integration** band. | (25-50] |
| 50 | <https://github.com/jwjacobson/jazz_repertoire/pull/65>  In pull request #65 of the jwjacobson/tune repository, the developer asked ChatGPT whether a Django view for listing tunes could cause an N+1 query issue:  rep\_tunes = RepertoireTune.objects.filter(player=user)  tunes = [tune.tune for tune in rep\_tunes]  ChatGPT confirmed the risk, explaining that if tune was a ForeignKey, accessing it in a loop would trigger additional queries. It recommended adding .select\_related('tune') to fetch related records in a single query.  During code review, a collaborator acknowledged the advice, linked the ChatGPT conversation, and noted the need for select\_related. However, the merged changes focused on renaming the rep\_tune field to tune, updating time handling with timezone.now, and simplifying random tune selection with .order\_by("?").first(). The N+1 prevention code (select\_related('tune')) was not implemented, leaving the performance issue in place.  Only the conceptual feedback from ChatGPT appeared in the discussion, without any direct adoption of its suggested code. | (25-50] |
| 51 | <https://github.com/jwjacobson/jazztunes/pull/65>  In pull request #65 of the jwjacobson/tune repository, the developer asked ChatGPT whether a Django view for listing tunes could cause an N+1 query issue:  rep\_tunes = RepertoireTune.objects.filter(player=user)  tunes = [tune.tune for tune in rep\_tunes]  ChatGPT confirmed the risk, explaining that if tune was a ForeignKey, accessing it in a loop would trigger additional queries. It recommended adding .select\_related('tune') to fetch related records in a single query.  During code review, a collaborator acknowledged the advice, linked the ChatGPT conversation, and noted the need for select\_related. However, the merged changes focused on renaming the rep\_tune field to tune, updating time handling with timezone.now, and simplifying random tune selection with .order\_by("?").first(). The N+1 prevention code (select\_related('tune')) was not implemented, leaving the performance issue in place.  Only the conceptual feedback from ChatGPT appeared in the discussion, without any direct adoption of its suggested code. | (25-50] |
| 52 | <https://github.com/mcagov/beacons/pull/539>  In pull request #539 of the uk-gov-mca/beacons-api repository, the developer sought ChatGPT’s advice on rolling back a Spring repository save when an exception is thrown outside the repository itself—specifically for non-database exceptions that occur after the save completes. ChatGPT recommended enabling declarative transaction management with @Transactional at the service layer, ensuring that exceptions are allowed to propagate (or are rethrown) so that Spring marks the transaction for rollback. It also suggested broadening rollback behavior when necessary by specifying rollbackFor.  In the final PR, the developer implemented service-level transactional boundaries and incorporated exception handling consistent with ChatGPT’s suggestions. Key service methods were annotated with @Transactional, some with rollbackFor = Exception.class, and exceptions thrown during Azure AD integration were rethrown as custom errors to trigger rollback. This mirrored ChatGPT’s core strategy, though much of the PR was dedicated to unrelated Azure AD integration and supporting classes.  Of the code directly relevant to ChatGPT’s advice, approximately 41% was integrated, reflecting partial adoption of the recommended rollback strategy within the broader scope of new feature development. This demonstrates how AI-generated guidance on transactional design can be selectively applied alongside significant unrelated changes. | (25-50] |
| 53 | <https://github.com/meta-llama/llama-recipes/pull/279>  In PR #279, the developer’s request to ChatGPT was for a simple Python requests client to send a POST request to a locally running Llama 2 API (vLLM on port 5000) with a prompt (“Who wrote the book Innovators dilemma?”), then print the result. ChatGPT returned a complete working example that exactly matched this usage pattern — specifying the endpoint, JSON payload, HTTP POST call, and conditional status code handling.  In the final merged code, the PR introduced comprehensive documentation and tutorials (demo\_apps/llama-on-prem.md, README.md, demo\_apps/README.md) for deploying and querying Llama 2 both with **vLLM** and **TGI** on-prem. Multiple query examples in the docs mirrored ChatGPT’s client logic almost verbatim:   * Same URL structure (http://localhost:5000/generate) * Same JSON payload fields (prompt, max\_tokens, temperature) * Same request/response flow via HTTP POST * Examples in both curl and Python client contexts that match the original ChatGPT snippet’s semantics.   While the PR expanded far beyond ChatGPT’s short script — covering multi-GPU configs, EC2 setup, OpenAI-compatible mode, LangChain integration, and TGI deployment — the original Python client request format from ChatGPT was integrated with little modification and appears multiple times in the documentation. This indicates **direct adoption of the suggested approach** for the querying mechanism, embedded within a broader system integration guide. | (25-50] |
| 54 | <https://github.com/nishiki-tech/nishiki-frontend/pull/223>  In PR #223, the team asked how to “detect that the outside of a specified element is clicked in React.” ChatGPT proposed a small, reusable pattern: (i) a useOutsideClick hook that attaches a mousedown listener to document, checks ref.current.contains(event.target), and cleans up on unmount; and (ii) a simple usage example via useRef in a component.  The merged code adds src/hooks/useOutsideClick.ts implementing exactly this logic (TypeScript-ized with RefObject<HTMLElement> and an optional callback) and wires it into the new SquareTextInput (useOutsideClick(inputWrapperRef, handleOutsideClick)) to close/submit/clear when clicking away. Aside from types and minor ergonomics (optional callback, comments), the approach mirrors ChatGPT’s prescription. | (25-50] |
| 55 | <https://github.com/osmosis-labs/osmosis-frontend/pull/1936>  In pull request #629, the developer sought to fix an issue where mobile browsers automatically zoomed into text inputs when focused in their React web app. ChatGPT suggested multiple solutions, including: (i) setting input font sizes to at least 16px to prevent zooming, (ii) ensuring the viewport meta tag was configured correctly, (iii) using JavaScript to dynamically disable zoom on focus, and (iv) applying CSS media queries for mobile adjustments. The final implementation adopted only the font-size fix for small screens via a CSS media query, setting input, textarea elements to 16px on devices with a max-width of 480px. This shows the developer selectively applied the simplest and most maintainable approach while discarding other proposed methods. | (25-50] |
| 56 | <https://github.com/poki/netlib/pull/50>  In pull request #50 of the repository, the developer encountered a PostgreSQL syntax error because DELETE … LIMIT 1 is not supported. They consulted ChatGPT for a workaround, and ChatGPT suggested using a Common Table Expression (CTE) to select a single row matching the condition with LIMIT 1, followed by a DELETE … USING to remove it while preserving the RETURNING clause. The final PR adopted the same CTE pattern but omitted some of the explanatory structuring and variable naming from ChatGPT’s example, integrating only the core logic needed for the fix. This case reflects selective adoption, where the developer took the essential part of the AI-generated fix but adapted it to fit the project’s coding style and existing query context. | (25-50] |
| 57 | <https://github.com/agentcoinorg/evo.ninja/pull/629>  In pull request **#629** of the polywrap/evo.ninja project, the developer asked how to stop mobile browsers from zooming when focusing text inputs. ChatGPT proposed several options: (1) set input font-size ≥16px, (2) ensure an appropriate viewport meta tag, and (3) optionally toggle viewport scaling via JavaScript. The merged fix implemented suggestion **(1)** almost verbatim by adding a media query that forces input, textarea { font-size: 16px !important; } on small screens. Review comments briefly considered the viewport-meta approach, but the team confirmed it was either already present or not the correct integration point for their Next.js setup. The final patch was a tiny CSS addition (+9 LOC), yet it exactly matches ChatGPT’s primary recommendation to eliminate iOS/Android auto-zoom on inputs. | (25-50] |
| 58 | <https://github.com/primer-io/primer-sdk-ios/pull/816>  In pull request #816 of the “Debug App” repository, the developer asked ChatGPT to “optimize” a small Swift MetadataParser that tries JSON first, then falls back to parsing KEY=VALUE lines. ChatGPT proposed a broader refactor aimed at speed/readability: use lazy sequences for the line pipeline, collapse maps/filters, switch to reduce(into:) for building the dictionary, parse booleans via a single lowercased() with a switch, and—importantly—try Int before Double for numeric parsing.  The merged code largely preserved the original structure (split → trim → filter → map → Dictionary(uniqueKeysWithValues:)), kept JSONSerialization with options: [], and still called lowercased() twice for the boolean checks. The only clear adoption from ChatGPT’s recommendations was swapping the numeric parsing order to check Int before Double. ChatGPT’s refactor served as review input, but the maintainer opted for minimal change—just the low-risk numeric parse tweak—while keeping the existing, readable pipeline intact. | (25-50] |
| 59 | <https://github.com/whatwg/html-build/pull/291>  In pull request #258 of the mobile fitness application repository, the developer sought clarification from ChatGPT on several JavaScript/React Native concepts, including the behavior of the logical && operator in JSX, the meaning of “truthy” values, and how to reset a typed useState object ({ [exerciseID: number]: boolean }). ChatGPT explained that && in JSX is commonly used for conditional rendering, defined “truthy” values in JavaScript, and showed how to reset the selectExercise state to an empty object. It also provided a complete example of integrating the reset function into a component.  The final merged PR introduced a new “Select Exercise by Muscle Group” page, implemented a FilteredExercises component that filtered exercises based on a selected muscle, and included touchable list items with dynamic background colors based on selection state. The TouchableOpacity and conditional styling logic closely mirrored the patterns in ChatGPT’s examples. While the developer extended the solution to work with IDs instead of objects and integrated it into a larger navigation flow, the core ideas—conditional rendering, dynamic styling based on state, and toggling selection—were directly aligned with ChatGPT’s guidance.  The structural and functional elements of ChatGPT’s examples were retained, but adapted to match the project’s broader state management and navigation context. | (25-50] |
| 60 | <https://github.com/Northeastern-Electric-Racing/FinishLine/pull/202>  In pull request #2232, the developer introduced a new identifier field to the Reimbursement\_Request table and implemented logic to backfill this field for existing records based on their dateCreated order. The ChatGPT discussion focused on SQL techniques for updating properties, handling multiple tables, and specifically generating sequential values using ROW\_NUMBER() for backfilling identifiers. ChatGPT provided guidance on using a CTE with a window function and joining it back to the table to update the column in the correct order.  In the final merged code, the developer applied a similar strategy in the Prisma migration script. They created a temporary table with sequential numbers ordered by dateCreated (and reimbursementRequestId as a tiebreaker), then updated the main table to set the identifier values before adding a unique constraint. This aligns closely with the ChatGPT-provided pattern, though the implementation also included schema changes (SERIAL column addition, Prisma model update, and test data adjustments) and integration into the seeding and transformer layers—areas not covered in ChatGPT’s responses.  The developer reused the key CTE-based sequential numbering approach from ChatGPT but adapted and extended it to match their migration tooling, schema constraints, and test requirements. This reflects substantial adoption of the core idea but with notable custom implementation details and additional steps beyond the AI’s example. | (50-75] |
| 61 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/229>  In pull request #2232, the developer migrated the ContactUsPage component from inline style objects and shared constants to a TailwindCSS-like utility class format, while retaining the original color scheme and layout. The ChatGPT discussion focused on converting React Native inline styles into utility-first class structures that mimic TailwindCSS, demonstrating how to replace repetitive style objects with utility classes for layout, spacing, and typography. ChatGPT’s code showcased the migration by applying className attributes with TailwindCSS utilities directly to each element while keeping color values consistent with the original code.  In the final merged code, the developer adopted the utility-based styling approach but refactored it to define common Tailwind class combinations in constants (viewStyles and textStyles) for reuse across the component. The implementation preserved the visual design and structural layout from ChatGPT’s suggestion, while adapting the usage pattern to fit the project’s preferred convention of centralized style variables.  This approach reflects alignment with ChatGPT’s recommendation, with only minor organizational adjustments for maintainability. | (50-75] |
| 62 | <https://github.com/UNLV-CS472-672/2024-S-GROUP3-Barbell/pull/238>  In pull request #238, the developer enhanced the muscle group selection UI by adding logic to unselect a muscle when it is already highlighted, grouping visually highlighted muscles according to the current schema, starting with an empty state to avoid errors, and introducing a "Next" button for navigation to the exercise selection page. Additionally, the developer expanded the exercise dataset in exercise.json.  The ChatGPT discussion focused on generating mock exercise data with a randomly assigned userId between 1 and 9 for each entry. ChatGPT provided a complete JSON file containing 60 exercise entries, each with a userId assigned in this range, ensuring consistent structure and randomized assignments.  In the final merged code, the developer directly integrated the generated mock data into exercise.json with minimal modifications, preserving ChatGPT’s provided structure and userId assignments. The rest of the PR involved unrelated UI and state management changes for the muscle selection page, which were not part of ChatGPT’s contribution. | (50-75] |
| 63 | <https://github.com/VOICEVOX/voicevox/pull/1844>  In this PR, the team refactored dialog-handling around “safe” save locations in the Electron main process. The ChatGPT discussion proposed restructuring the logic into a loop that keeps prompting until a valid path is chosen, extracting two helpers—one to determine whether a path is unsafe and another to show a warning dialog—and returning early on cancel. It also recommended simplifying the control flow and clarifying the path-safety check.  In the merged code, the developer implemented a function with the same core structure (a while (true) loop that re-prompts, an unsafe-path check, and a warning dialog), but adapted details to the project’s needs: the helpers were inlined as local functions, the unsafe-path rule added a ..${path.sep} branch, the warning dialog included the app name and returned "retry"/"forceSave", and the function was integrated across multiple IPC handlers (SHOW\_\*\_SAVE\_DIALOG and directory selection). ESLint was updated to allow constant-condition loops, and the new safe-save flow was wired into the preload, sandbox, UI, and store layers.  This reflects substantial adoption of ChatGPT’s refactoring approach (looping prompt, safety check, warning dialog, early returns), with project-specific enhancements and broader integration work beyond the original snippet. | (50-75] |
| 64 | <https://github.com/aleph-im/aleph-vm/pull/460>  In this pull request, the developer added a patch\_datetime\_now fixture to tests for patching datetime.now() and datetime.utcnow() to return a fixed timestamp. In the ChatGPT conversation, the developer provided an existing fixture and requested a docstring. ChatGPT responded with an extended version containing a descriptive docstring that explained the fixture’s purpose, parameters, return type, and usage example. The AI’s suggestion preserved the fixture’s existing logic, making only a documentation addition.  In the final merged code, the developer included a patch\_datetime\_now fixture in the new test suite, with a docstring that briefly described its purpose and functionality. While the implemented docstring was more concise than ChatGPT’s detailed example, it still reflected the same intent—clarifying the fixture’s role in mocking current datetime values for tests. The integration aligns with ChatGPT’s contribution in both placement and purpose, though the wording and level of detail were altered to match the repository’s style and tone.  Since the developer reused the key docstring concept from ChatGPT but condensed it, the integration rate falls in the **50–75% range**. This shows substantial adoption of the AI’s idea with style adjustments. | (50-75] |
| 65 | <https://github.com/carrot-stick/design-system/pull/12>  In pull request #12 of the carrot-stick design-system project, the author asked (in Korean) how to enforce two JSX conventions via tooling and auto-formatting:  (i) use plain double quotes for string props instead of {'…'}, and  (ii) use the boolean-prop shorthand (error instead of error={true}).  ChatGPT proposed an ESLint+Prettier setup: add jsx-quotes: "prefer-double" and react/jsx-boolean-value: "never" in ESLint, keep JSX double quotes via Prettier (e.g., jsxSingleQuote: false), and integrate eslint-config-prettier/eslint-plugin-prettier with example configs and install commands.  The merged PR shows a broad formatting sweep and config churn (38 files touched) plus a Prettier config migration: a new **.prettierrc** JSON (9 lines) replaced **.prettierrc.js**, and repository formatting changes landed across components (Buttons, Flex, Input, Select, new Modal, etc.). The dependency graph also contains the Prettier/ESLint plugins noted in the advice. However, the diff excerpts do not show the two **specific ESLint rules** (jsx-quotes, react/jsx-boolean-value) being added to an ESLint config, and the new .prettierrc does not explicitly carry a JSX-quotes setting (it relies on defaults). In short, the team adopted the **Prettier-centric** portion of the guidance and applied project-wide formatting, while **not clearly integrating** the precise ESLint rules that would guarantee the boolean-shorthand and JSX-quote behaviors.  Overall, this reflects **partial adoption** of ChatGPT’s recommendations—tooling alignment and formatting were applied, but the exact linter rules weren’t obviously added. | (50-75] |
| 66 | <https://github.com/deso-protocol/core/pull/875>   In this exchange, the developer asked whether reading and writing a shared pointer in Go requires synchronization, and whether a racy read could yield a “random” pointer. ChatGPT’s first answer recommended using a mutex, emphasizing memory-visibility guarantees. When the user pressed on pointer-copy semantics, ChatGPT clarified that (a) unsynchronized access is a data race and therefore incorrect per Go’s memory model, but (b) word-sized pointer reads/writes are effectively atomic on typical platforms, so a racy read would observe some real, previously written value (old or new), not an arbitrary bit pattern. The follow-up summary against the official “Go Memory Model” was directionally right about DRF-SC, the need for synchronization, and that races are errors, though it could have been tighter by explicitly highlighting the model’s guarantee for single-machine-word reads (“each read must observe a value written by a preceding or concurrent write”) and the ban on “out-of-thin-air” results.  The subsequent pull request is a broad PoS/consensus update touching config flags, event-loop wiring, QC validation helpers, mempool/UTXO logic, and new collection utilities (e.g., a ConcurrentList[T] guarded by sync.RWMutex, equality helpers, and slice/map utilities). There is no evidence that ChatGPT authored any of the committed code; at most, the conversation appears to have informed concurrency posture in small corners (e.g., introducing an explicitly synchronized list and copying slices on read). Because the PR spans many subsystems and the chat produced no concrete patch, the integration of AI-generated lines is best categorized as minimal.  Impact and takeaways  • Correctness: The conversation reinforced that unsynchronized pointer reads/writes constitute a data race even if the pointer-sized copy is atomic; code that depends on “it’s probably fine” is non-portable and brittle.  • Practical guidance: Prefer explicit synchronization (mutexes, channels) or atomic.Pointer/atomic.Value for read-mostly pointer handoffs. If readers also dereference and act on the pointed-to data while writers replace it, guard both the pointer and the object’s lifetime protocol.  • In the PR, concurrency-sensitive helpers moved toward safer patterns (e.g., guarded container with snapshot-on-read). That aligns with the conversation’s thrust but does not reflect direct code import from ChatGPT. | (50-75] |
| 67 | <https://github.com/faker-js/faker/pull/2405>  In pull request #2405 of the repository, the developer aimed to implement a CI safeguard that would **block merging a PR if it contained either of two labels**—"do NOT merge yet" or "s: on hold". They consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/f09f38e5-f541-4f98-9483-e183f5650398}{ChatGPT](https://chat.openai.com/share/f09f38e5-f541-4f98-9483-e183f5650398%7D%7BChatGPT) conversation link} -- Accessed: 2025-02-17}, which suggested a GitHub Actions workflow triggered on pull\_request label-related events. The workflow iterates over a predefined array of blocking labels, extracts PR labels from the event payload via jq, and exits with a non-zero status if a match is found, thereby preventing merges.  The final merged workflow, .github/workflows/check-mergable-by-label.yml, closely followed ChatGPT’s proposed approach:  (i) it defined the same blocking label list,  (ii) parsed labels from $GITHUB\_EVENT\_PATH using jq, and  (iii) exited with status 1 when a match occurred.  Only minimal adaptations were made—the trigger list was expanded to include PR lifecycle events beyond labeling, and the job/step names were customized. | (50-75] |
| 68 | <https://github.com/formbricks/formbricks/pull/2351>   In a pull request from the formbricks/formbricks repository, the developer sought ChatGPT’s advice on refactoring a VideoSettings React component to improve maintainability and readability. ChatGPT recommended:  (i) extracting event handlers and URL parsing logic into utility functions,  (ii) introducing a custom useVideoSettings hook to encapsulate state and handlers,  (iii) wrapping event handlers with useCallback for performance, and  (iv) simplifying conditional rendering.  The developer integrated the spirit of these recommendations by creating a new shared utility module, videoUpload.ts, that centralizes YouTube/Vimeo/Loom URL validation, ID extraction, and parsing logic. They also introduced a reusable QuestionMedia component with a LoadingSpinner to consolidate and enhance media display logic across multiple survey components. These changes improved consistency, reduced duplication, and propagated video support throughout the application.  However, the specific custom hook proposed by ChatGPT was not implemented, and only limited useCallback usage appeared. The provided refactored VideoSettings example was not integrated verbatim, suggesting the team favored project-wide structural improvements over direct adoption of ChatGPT’s sample code.  This reflects substantial uptake of the architectural and utility-driven refactoring ideas, while selectively omitting certain implementation details to better match the existing codebase’s patterns. | (50-75] |
| 69 | <https://github.com/g12-4soat/techlanches-lambda-auth/pull/1>   In a pull request from the formbricks/formbricks repository, the developer sought ChatGPT’s advice on refactoring a VideoSettings React component to improve maintainability and readability. ChatGPT recommended:  (i) extracting event handlers and URL parsing logic into utility functions,  (ii) introducing a custom useVideoSettings hook to encapsulate state and handlers,  (iii) wrapping event handlers with useCallback for performance, and  (iv) simplifying conditional rendering.  The developer integrated the spirit of these recommendations by creating a new shared utility module, videoUpload.ts, that centralizes YouTube/Vimeo/Loom URL validation, ID extraction, and parsing logic. They also introduced a reusable QuestionMedia component with a LoadingSpinner to consolidate and enhance media display logic across multiple survey components. These changes improved consistency, reduced duplication, and propagated video support throughout the application.  However, the specific custom hook proposed by ChatGPT was not implemented, and only limited useCallback usage appeared. The provided refactored VideoSettings example was not integrated verbatim, suggesting the team favored project-wide structural improvements over direct adoption of ChatGPT’s sample code.  This reflects substantial uptake of the architectural and utility-driven refactoring ideas, while selectively omitting certain implementation details to better match the existing codebase’s patterns. | (50-75] |
| 70 | <https://github.com/hackforla/CivicTechJobs/pull/477>  In a PR to fix a failing MkDocs deployment workflow, the run logs showed ModuleNotFoundError: No module named 'setuptools' when running setup.py. ChatGPT recommended (i) ensuring setuptools (and wheel) are installed, (ii) pinning Python via actions/setup-python (and updating to @v4), (iii) removing duplicate pip install steps, (iv) simplifying/removing cd chains, and (v) invoking mkdocs gh-deploy with an explicit --config-file.  The merged PR implemented essentially all of this: switched to actions/setup-python@v4, pinned python-version: "3.9", added an “Install dependencies” step with pip setuptools wheel, removed the duplicate installs, dropped the cd hops, and used mkdocs gh-deploy --config-file mkdocs/mkdocs.yml. The workflow now installs prerequisites before running python mkdocs/mkdocs-config/setup.py develop, addressing the missing setuptools error. | (50-75] |
| 71 | <https://github.com/hackforla/CivicTechJobs/pull/477>  In pull request #241 of the nbd-wtf/nip19 repository, the developer aimed to reduce duplication in a TypeScript discriminated union mapping NIP-19 prefixes (npub, nprofile, etc.) to their respective payload types. They consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/f09f38e5-f541-4f98-9483-e183f5650398}{Link](https://chat.openai.com/share/f09f38e5-f541-4f98-9483-e183f5650398%7D%7BLink) to ChatGPT conversation} -- Accessed: 2025-02-17}, which proposed consolidating the type definitions by:  (i) defining a single PrefixMap object that maps keys to their data types,  (ii) introducing a generic DecodeValue<P> type that automatically infers the data type from PrefixMap[P], and  (iii) creating the DecodeResult type via a mapped type indexed over keyof PrefixMap to produce the discriminated union without repetition.  The final merged code adopted this approach nearly verbatim, renaming PrefixMap to Prefixes and adding template literal types for stricter typing of encoded and decoded strings (e.g., `npub1${string}`). These additions enhanced type safety but did not alter the underlying structure recommended by ChatGPT.  This case demonstrates how ChatGPT’s mapped type and lookup type approach can be directly applied to simplify complex union definitions while leaving room for developer-led type refinements. | (50-75] |
| 72 | <https://github.com/netdata/netdata/pull/15455>  Pull request #15455 of the netdata/netdata repository improved the "What's New" section of the project’s README by reordering the Markdown table columns to place **“What”** and **“Description”** first, followed by **“When”** and **“Status”**. The contributor asked ChatGPT\footnote{\href{[https://chat.openai.com/share/...}{Conversation](https://chat.openai.com/share/...%7D%7BConversation) link} — Accessed: 2025-02-17} which ordering—Option A (What | Description | When | Status) or Option B (When | Status | What | Description)—would be more effective. ChatGPT recommended **Option A**, arguing that users typically care most about what the feature is and its description before seeing scheduling or status details, and that this order is more intuitive for left-to-right reading and better for mobile viewing.  The maintainer adopted this advice directly:  (i) reordered the table to place “What” and “Description” first,  (ii) retained “When” and “Status” as the last two columns, and  (iii) updated all feature links to point to the relevant agent release notes.  The case shows how ChatGPT’s input can directly shape user-facing documentation layout when maintainers agree on the UX rationale. | (50-75] |
| 73 | <https://github.com/nylas/nylas-python/pull/279>  Pull request #279 of the nylas/nylas-python repository added a new **free-busy** endpoint to the SDK and required a response model that can contain either availability data or error objects. The author asked ChatGPT\footnote{\href{[https://chat.openai.com/share/51f3aa63-d8aa-4ff7-aca1-608fcf9ab9ee}{Conversation](https://chat.openai.com/share/51f3aa63-d8aa-4ff7-aca1-608fcf9ab9ee%7D%7BConversation) link} — Accessed: 2025-02-17} how to express the JSON as Python 3 dataclasses. ChatGPT proposed four dataclasses—TimeSlot, FreeBusy, Error, and a top-level ResponseData with List[Union[FreeBusy, Error]]—capturing (i) separate models per object, and (ii) a **Union** at the top level to represent mixed lists.  In review, maintainers asked to (a) **rename** Error to FreeBusyError for clarity and (b) keep a bespoke GetFreeBusyResponse (rather than the SDK’s paginated ListResponse, since free-busy isn’t paginated). The merged patch introduced FreeBusyError, TimeSlot, FreeBusy, and GetFreeBusyResponse (still using List[Union[FreeBusy, FreeBusyError]]), plus a GetFreeBusyRequest TypedDict and the calendars.get\_free\_busy(...) resource method wired to return Response[GetFreeBusyResponse]. Minor divergences included adoption of dataclasses\_json (not mentioned by ChatGPT) and rejecting the paginated wrapper.  Of the ~25 LOC ChatGPT suggested for the models, ~15 LOC (≈60%) were integrated—squarely **upper-median**—with edits focused on naming (FreeBusyError), concrete SDK fit (GetFreeBusyResponse), and avoiding pagination semantics. | (50-75] |
| 74 | <https://github.com/pokt-foundation/pocket-go/pull/93>  pull request #93 in the pokt-foundation/pocket-go repository introduced a mechanism to capture and propagate HTTP status codes more accurately during relay requests. The change addressed limitations in the existing extractStatusFromResponse function, which previously defaulted to returning "200" when no known error patterns were matched. This behavior risked masking unknown or unexpected outcomes as successes.  After the developer consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/111342f3-0045-4e71-b24b-4e2b8822dd25}{Example](https://chat.openai.com/share/111342f3-0045-4e71-b24b-4e2b8822dd25%7D%7BExample) link to ChatGPT conversation} -- Accessed: 2025-02-17}, the model suggested several improvements: (i) moving regex compilation outside the function to avoid repeated processing, (ii) changing the default status code to an “unknown” indicator rather than implicitly returning 200, and (iii) adding clearer documentation. ChatGPT also proposed a refactored function with globally scoped regex patterns and more explicit handling of non-matched cases.  The initial ChatGPT-generated patch (17 LOC) implemented these suggestions directly in the function body. However, the integrated code took a broader approach — moving regex compilation into an init() function, introducing a defaultStatusCode constant set to 202 (instead of “unknown”), expanding regex coverage to detect 200 OK explicitly, and adding error-to-status mapping for known network issues. These changes were applied across multiple files (provider.go, relay.go, and test suites), resulting in a more comprehensive error propagation system.  This case shows how ChatGPT’s feedback influenced both the structure (precompiled regexes) and semantics (avoiding misleading defaults) of the final implementation, while the development team adapted these ideas to align with internal API conventions and project constraints. | (50-75] |
| 75 | <https://github.com/sCrypt-Inc/scryptTS-docs/pull/209>  Pull request #209 of the sCrypt-Inc repository updated the documentation for programmatic webhook registration, replacing a large inline JSON artifact with code that dynamically reads the artifact from a file before sending it in an HTTP request. To implement this, the developer consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/79885c38-345b-4d2f-9f4f-d947562490b8}{Example](https://chat.openai.com/share/79885c38-345b-4d2f-9f4f-d947562490b8%7D%7BExample) link to ChatGPT conversation} -- Accessed: 2025-02-17}, asking how to read a JSON file in Node.js using await. ChatGPT suggested using the fs.promises API with readFile and JSON.parse inside an asynchronous helper function, wrapped in a try/catch block, and demonstrated its use in a main() function that awaited the result before processing the data.  In the final merged change, the developer created a fetchArtifactFromFile helper mirroring ChatGPT’s approach: importing fs.promises, reading the file asynchronously, parsing it as JSON, and returning the object. This function was then invoked in the webhook registration example to supply the artifact field dynamically. The integration retained the key control flow, error handling, and API usage from ChatGPT’s example, with minor deviations such as including an unused util import and omitting the Node version note for fetch.  The developer adopted the core implementation pattern directly but tailored the surrounding context to fit the documentation example and project conventions. The unused import and environment assumptions slightly reduced alignment but did not materially affect the functional match. | (50-75] |
| 76 | <https://github.com/swarmion/swarmion/pull/678>  In pull request [#678](https://github.com/swarmion/swarmion/pull/678) of the swarmion/swarmion repository, the developer aimed to add clear documentation for the release.sh script used to automate releases in the project. The developer consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/7538b618-c08d-45b7-a4ed-bb168e9c1eb0}{ChatGPT](https://chat.openai.com/share/7538b618-c08d-45b7-a4ed-bb168e9c1eb0%7D%7BChatGPT) conversation link} — Accessed: 2023-07-27} by providing the entire Bash script and requesting concise, bullet-pointed documentation with usage instructions and examples.  ChatGPT’s response included:   1. A detailed description of the script’s functionality in nine sequential steps. 2. A “Usage” section showing required and optional arguments. 3. Two usage examples with commands. 4. Additional notes on assumptions and recommended execution environments.   The final PR integrated **almost all** of ChatGPT’s suggested text into the CONTRIBUTING.md file with only minor adjustments:   * The wording was slightly tailored to fit the repository’s style (e.g., changing “for a project” to “for swarmion”). * The prerequisites section was added manually by the developer to describe npm organization access. * The note on “secure execution environment” from ChatGPT’s output was omitted.   The patch consisted of approximately 43 lines of generated documentation, and 40 lines (≈93%) were incorporated with minimal rephrasing.  This case demonstrates a **high level of AI-generated documentation integration** where the ChatGPT output served as the direct foundation for the PR content, requiring only small contextual edits and project-specific additions before merging. | (75-100] |
| 77 | <https://github.com/g12-4soat/tech-lanches/pull/66>  In pull request [#66](https://github.com/g12-4soat/techlanches/pull/66) of the g12-4soat/techlanches repository, the developer sought to create a complete set of Kubernetes YAML manifests for deploying a .NET 6–based application consisting of an API, a background worker, and a SQL Server database. The deployment needed to include a namespace, secrets, config maps, deployments, services, persistent storage, horizontal pod autoscalers (HPAs), and a unified shell script for applying all resources.  To expedite this process, the developer provided ChatGPT\footnote{\href{[https://chat.openai.com/share/fdf5e03a-c4b0-498a-968c-8585714840d4}{ChatGPT](https://chat.openai.com/share/fdf5e03a-c4b0-498a-968c-8585714840d4%7D%7BChatGPT) conversation link} — Accessed: 2023-11-20} with detailed specifications for each required component. ChatGPT responded with:   1. Fully written YAML definitions for each Kubernetes resource, including separate files for API, worker, and database components. 2. Probes, namespace creation, config map linking, and secret integration for database connection strings. 3. HPAs for both the API and worker with scaling rules based on CPU and memory utilization. 4. A shell script (apply-all.sh) containing the sequence of kubectl apply commands to deploy everything at once. 5. Explanations for why executable permissions (chmod +x) are necessary in Unix-like systems, plus handling notes for different OS environments.   The final merged PR contained Kubernetes YAML files that closely mirrored the ChatGPT-generated manifests, with only modest adjustments:   * The developer added **StatefulSet** definitions and load balancer services for SQL Server following Microsoft’s Kubernetes best practices. * Certain values (e.g., image repository tags, storage configurations, and probes) were adapted to match project conventions. * The apply-all.sh script was simplified to a single recursive kubectl apply -f . --recursive rather than listing each file individually.   This case illustrates a **high integration scenario** where ChatGPT served as the primary author of the Kubernetes deployment specifications, enabling rapid infrastructure setup with only targeted refinements from the developer to align with operational and architectural guidelines. | (75-100] |
| 78 | <https://github.com/VyProductions/SeniorDesign/pull/57>  In pull request [#57](https://github.com/VyProductions/YourRepo/pull/57) of the VyProductions repository, the developer aimed to refactor a Python program that calculated change in bills and coins. The original implementation contained eight separate get\_\* functions—one for each bill or coin type—which resulted in a lengthy, repetitive codebase. Additionally, the program lacked robust error handling for non-numeric input and improper monetary formatting.  To address these issues, the developer consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/968d797a-0ef8-4644-91b9-70d1a3d0c016}{ChatGPT](https://chat.openai.com/share/968d797a-0ef8-4644-91b9-70d1a3d0c016%7D%7BChatGPT) conversation link} — Accessed: 2024-02-14}, which recommended:   1. **Consolidating all denomination-specific functions** into a single, reusable get\_change function that takes the amount and the denomination value as arguments, returning both the count of that denomination and the updated remaining amount. 2. **Replacing repetitive function calls** with iteration over a list of tuples containing the denomination value and its display name, simplifying output formatting and maintenance. 3. **Adding robust error handling**, including:     * A try/except block to catch invalid (non-numeric) inputs.    * An additional check to ensure the monetary value has up to two decimal places for proper currency formatting.    * A validation step to reject amounts greater than $100. 4. **Reducing code length** from 115 lines to 46 lines without losing functionality, significantly improving readability and maintainability.   The final merged PR retained nearly all elements of ChatGPT’s refactored design, with the structure, logic, and error checks directly aligned to the suggested solution. Only minor adaptations were made, such as keeping the naming conventions and error messages consistent with the project’s style.  This case represents a **high-integration scenario** where ChatGPT’s proposed refactor served as the main blueprint for the final merged code, substantially streamlining logic while introducing best practices for error handling and code organization. | (75-100] |
| 79 | <https://github.com/VyProductions/SeniorDesign/pull/51>  In pull request [#51](https://github.com/VyProductions/YourRepo/pull/51) of the VyProductions repository, the developer sought to refactor a Python program that calculated force using the formula *F = m × a*. The original code used **two separate while loops**—one for validating mass and another for acceleration—which led to redundant logic and unnecessary repetition.  The developer consulted ChatGPT\footnote{\href{[https://chat.openai.com/share/dc460461-5653-44b6-b996-3e00ace11f98}{ChatGPT](https://chat.openai.com/share/dc460461-5653-44b6-b996-3e00ace11f98%7D%7BChatGPT) conversation link} — Accessed: 2024-02-13}, which suggested:   1. **Merging the two while loops into a single loop** that prompts for both inputs each iteration. 2. **Validating both inputs simultaneously** by checking if mass > 0 and acceleration > 0 before breaking the loop. 3. Retaining the same program flow—loop until valid inputs are provided, then compute and display the force.   The final merged PR directly implemented ChatGPT’s proposed solution with minimal alteration. The resulting code was reduced to **8 lines**, maintaining the same functionality while eliminating redundancy. The logic, structure, and break condition matched the AI-generated recommendation exactly.  This represents a **high-integration case** where ChatGPT’s refactoring was adopted almost verbatim, producing cleaner, more concise code without affecting behavior. | (75-100] |
| 80 | <https://github.com/EthicApp-Development/ethicapp-main/pull/301>  In pull request **[#301]** of the EthicApp-Development/ethicapp repository, the team undertook a large schema rework for the **api-v2** backend, introducing many Sequelize models and migrations and aligning table/attribute names with Sequelize conventions. Ahead of this, the developer consulted ChatGPT (conversation in Spanish) for guidance on **how Sequelize handles models vs. migrations**, how to **generate both via sequelize-cli model:generate**, how to **author incremental migrations** (e.g., addColumn, FKs), and how to **create an N–N join table** with proper foreign keys and (optionally) composite primary keys. **What ChatGPT proposed**  1. Keep **models and migrations separate**; use sequelize-cli to scaffold. 2. Use sequelize model:generate --name X --attributes ... to create **model + migration** together for entities like User (name,email). 3. For incremental changes, use sequelize migration:generate and methods like addColumn, with **FK options** (references, onUpdate/onDelete). 4. For **many-to-many**, create a join table with two FK columns, **CASCADE** rules, and (optionally) a **composite primary key** across both FKs.  **What the PR adopted from ChatGPT**  * Introduced/standardized **Sequelize CLI structure** (e.g., .sequelizerc) and a full set of **timestamped migrations** under api/v2/migrations. * Created foundational tables via migrations: users, sessions, questions, teams, plus **join tables** sessions\_users and teams\_users with **references** to parent tables—matching the N–N guidance. * Followed the separation-of-concerns pattern (models define shape/associations; **migrations** are the vehicle for schema changes). * Performed wide **renaming to match conventions** (pluralized snake\_case tables), consistent with ChatGPT’s emphasis on Sequelize conventions.  **Where the PR diverged or refined**  * The join-table migrations **did not** implement the **composite primary key** pattern suggested; instead, at least one join table uses an auto-increment id, and **CASCADE** rules were not consistently specified. * Field sets and naming differ from the simple examples (e.g., users includes rut, pass, mail, sex, role, etc.), reflecting project-specific needs rather than ChatGPT’s minimal samples. * The PR includes substantial **Express routing and API wiring** (controllers/routers, JWT middleware)—beyond the scope of the ChatGPT schema guidance.  **Verdict** This is an **Upper-median integration** case: ChatGPT’s **approach and scaffolding strategy** (CLI usage, migration-first changes, FK-based join tables, incremental migrations) were clearly followed, and **key structural recommendations** appear in the merged code. However, several **implementation details** (composite PK on join tables, explicit onDelete/onUpdate policies) were adapted or omitted. | (75-100] |
| 81 | <https://github.com/hacktoberfestkorea/hacktoberfestkorea/pull/241>  In pull request **#241** of the hacktoberfestkorea repository, the contributor wanted a GitHub Action that automatically labels new issues with **hacktoberfest**. They asked ChatGPT for a workflow to do this, and ChatGPT proposed a minimal action that:   * Triggers on issues: opened * Uses actions/github-script@v5 * Calls github.rest.issues.addLabels(...) with labels: ['hacktoberfest'] * Authenticates via ${{ secrets.GITHUB\_TOKEN }}   The merged PR adds .github/workflows/add\_hacktoberfest\_tag.yml containing **the same 22-line workflow** as suggested, line-for-line (name, trigger, job, step, script, and token usage all match). No additional logic or structure was introduced. | (75-100] |
| 82 | <https://github.com/alshedivat/al-folio/pull/2059>  In a pull request to the alshedivat/al-folio repository, the developer asked ChatGPT to enhance a Jekyll plugin so it could fetch posts not only from RSS feeds but also from manually specified URLs. The request included adding HTML parsing to extract titles, descriptions, and content, handling publication dates more robustly, and improving logging output  **ChatGPT’s Contributions**  ChatGPT proposed:   1. Splitting the plugin logic into helper methods (fetch\_from\_rss, process\_entries, fetch\_from\_urls, create\_document). 2. Adding Nokogiri for HTML parsing and Time handling for publication dates. 3. Supporting a new YAML config format with a posts list, each containing a url and published\_date. 4. Implementing parse\_published\_date to handle both string and Date types from YAML. 5. Changing logging from p to puts for cleaner output.   **Adoption in PR**  The merged code integrates nearly all of ChatGPT’s suggested structure and logic verbatim:   * All new helper functions were added exactly as proposed. * The HTML parsing, metadata extraction, and date handling logic matches ChatGPT’s code. * The updated \_config.yml format was implemented as described. * The logging improvement was applied.   A small addition was made by the developer to add return if xml.nil? in the RSS handler, but this did not replace or alter ChatGPT’s work.  **Integration Level & Reasoning**  ChatGPT’s patch introduced around 72 lines of functional changes; all of these were retained in the final PR, making this a **~100% integration**. The near-direct adoption happened because the AI’s output aligned perfectly with the project’s structure and coding style, requiring no rewrites.  **Summary**  This is a clear case where ChatGPT’s solution was production-ready, with the developer applying it almost exactly as delivered. The PR shows how AI-assisted code generation can serve as a drop-in solution when the requirements are well understood and the generated code matches project conventions. | (75-100] |
| 83 | <https://github.com/netdata/netdata/pull/15485>  In a pull request to the netdata/netdata repository, the developer asked ChatGPT about the behavior of a shell snippet for locating the md5sum binary and how it would affect a system ID–generation block. ChatGPT explained that if md5sum was missing, $MD5\_PATH would be empty, and the logic would fall back to alternative methods (e.g., uuidgen or "null"). The discussion also detailed each conditional branch, clarifying when the machine-id or DBus ID hashing steps would run and when they would be skipped.  **ChatGPT’s Contributions**  From the conversation, ChatGPT provided:   1. An explanation of the MD5\_PATH detection logic and its failure behavior. 2. A walkthrough of the conditional SYSTEM\_DISTINCT\_ID assignment, including all fallback paths. 3. Recommendations to explicitly check $MD5\_PATH before using it, ensuring the script avoids errors when md5sum is missing.   **Adoption in PR**  The final merged PR incorporated some—but not all—of these recommendations:   * $MD5\_PATH is now set using the multi-method detection approach (which, command -v, type). * All machine ID and DBus ID branches now guard execution with [ -n "$MD5\_PATH" ], as ChatGPT suggested. * The fallback to uuidgen and "null" remains, consistent with ChatGPT’s logic.   However, the code from the conversation was not copied verbatim:   * The PR includes additional logic in other parts of the script not discussed with ChatGPT. * The exec <&- 2>&- usage and logging/variable naming details differ slightly from ChatGPT’s draft. * The conversation was largely advisory; the PR’s implementation was a tailored adaptation rather than a direct lift of code.   **Summary**  This case illustrates a scenario where ChatGPT’s conceptual advice and safety checks were adopted, but the implementation details were customized by the developer. The AI’s main impact was in reinforcing safe $MD5\_PATH checks before attempting to hash a machine ID, ensuring graceful fallbacks when md5sum is missing. | (75-100] |
| 84 | <https://github.com/dantol29/webserver/pull/147>  In a web server project pull request, the developer reworked the CGI handling pipeline while improving request and response processing. The changes replaced the legacy Environment structure with a more precise MetaVariables model, introduced AResponseHandler to replace ARequestHandler, refined routing logic in Router to detect CGI by file extension, and normalized static versus dynamic content handling. The update also streamlined request reading in Connection::readRequestHeadersAndBody, improved static file delivery in StaticContentHandler, and added a dedicated webroot with example CGI scripts and a custom 404.html page.  Before implementing these changes, the developer consulted ChatGPT to verify C++ return value optimization details and to review the proposed CGI code. ChatGPT’s contributions included:   1. Fixes to the exec call path — ensuring properly sized argv arrays and safely joining PATH\_TRANSLATED with SCRIPT\_NAME. 2. Improved CGI header parsing — skipping stray HTTP status lines from scripts, respecting Status: headers, correctly stripping the \r\n\r\n separator, and ensuring Content-Length is present. 3. Better environment and I/O handling — building the envp array without memory leaks and correctly passing request bodies to CGI scripts via stdin pipes. 4. Routing and static content polish — normalizing “/” requests to /index.html, checking for file readability, and returning structured 404 responses.   The ChatGPT-generated patch consisted of approximately **210 LOC**, of which **192 LOC (91.4%)** were integrated directly with minimal modifications. A few ideas, such as adding CGI timeouts and broader security measures, were noted for future work. The high percentage of adoption reflects how closely the AI’s suggestions aligned with the project’s needs, fitting seamlessly into the final merged version. | (75-100] |
| 85 | <https://github.com/PrivateBin/PrivateBin/pull/1134>  In pull request #1134 of the PrivateBin/PrivateBin repository, the maintainer introduced multiple GitHub issue templates to better categorize incoming reports, including a dedicated GitHub Discussion category form for "Question and support" inquiries. This change replaced the single generic issue template with several structured templates, aiming to improve triage efficiency, nudge reporters toward more complete submissions, and redirect general support questions away from the bug tracker.  Before committing these changes, the contributor used ChatGPT to convert an existing Markdown-based support request template into GitHub’s YAML discussion category form syntax. The AI-generated output included:   1. **Title and labels configuration** – "title": "[Question and support] " and labels: ["question/support"]. 2. **Structured body definition** – Markdown introduction, confirmation checkboxes, and sectioned inputs for reproduction steps, observed/expected behavior, additional information, and environment details. 3. **Field variety** – Multiple input types (markdown, checkboxes, textarea, input, radio) to collect both structured and free-form data from users.   The ChatGPT-suggested YAML patch contained approximately **62 LOC**, of which **62 LOC (100%)** were integrated into the final .github/DISCUSSION\_TEMPLATE/q-a.yml file without modification. This resulted in a perfect adoption rate.  The direct one-to-one match between ChatGPT’s output and the merged code indicates that the AI-generated form aligned exactly with the maintainer’s requirements, requiring no post-generation adjustments. This case demonstrates how ChatGPT can be used effectively to translate an existing unstructured template into GitHub’s structured form syntax with full production readiness. | (75-100] |
| 86 | <https://github.com/UNLV-CS472-672/2024-S-GROUP1-Roadwatch/pull/105>  In pull request #105 of the UNLV-CS472-672/2024-S-GROUP repository, backend functionality was added to manage markers on a map, including endpoints to create (POST), retrieve (GET), and delete (DELETE) marker entries from the database. These endpoints integrated with a MongoDB model (Marker) and were protected with authentication middleware.  The contributor used ChatGPT to enhance the existing code by adding inline comments and improving documentation for clarity. ChatGPT’s contributions included:   1. **Endpoint explanations** – Clear descriptions above each controller function (getMarkers, saveMarker, deleteMarker). 2. **In-line process comments** – Step-by-step annotations inside each function explaining data retrieval, validation checks, database operations, and response handling. 3. **Error handling notes** – Documentation on how errors are caught and the HTTP responses returned.   The ChatGPT-suggested patch was approximately **53 LOC**, and the final merged version retained **49 LOC** of these (92.45% integration). The minor differences came from the maintainer adding more descriptive error messages (e.g., "Could not get markers" instead of "Internal server error") and small refactoring for project style consistency.  This case illustrates a near-complete adoption of ChatGPT’s documentation enhancements, showing that AI-generated explanatory comments can be production-ready with minimal human adjustment, while also benefiting from small project-specific refinements. | (75-100] |
| 87 | <https://github.com/open-learning-exchange/myplanet/pull/2229>  In pull request #934 of the ole-planet/myplanet repository, the developer focused on improving the French localization of the Android application. This work involved translating and refining strings.xml entries and string arrays, ensuring consistency across UI elements and system messages.  Before implementation, the developer consulted ChatGPT to assist in translating numerous English strings and arrays into French. ChatGPT provided direct translations for UI labels (e.g., “Sign In” → “Se connecter”), descriptive text, error messages, and array items such as levels, subjects, and resource types. The translations also covered subtle details, like adding proper French punctuation, using accents, and adapting terms for cultural clarity (e.g., “Manager Login” → “Connexion en tant que gestionnaire”).  In the final merged PR, the majority of ChatGPT’s suggested translations were integrated verbatim, with only minor stylistic adjustments (such as rewording “Flow Video Player” to “Lecteur vidéo de flux” or refining the tone of certain messages). Since the core content, structure, and word choice closely mirrored ChatGPT’s output, the integration rate is estimated at above 90%, placing it in the high adoption category.  This case shows how ChatGPT can serve as an efficient language localization assistant, providing high-quality translations that require minimal human post-editing, thus accelerating the internationalization process for software projects. | (75-100] |
| 88 | <https://github.com/open-learning-exchange/myplanet/pull/2230>  In pull request #2230 of the ole-planet/myplanet repository, the developer introduced comprehensive Nepali translations for the application's interface, replacing English UI text with localized equivalents. The update targeted strings.xml for the values-ne directory, ensuring that menus, labels, messages, prompts, and array items reflected culturally and linguistically appropriate Nepali usage.  To prepare these translations, the developer used ChatGPT (links provided in the PR description) to generate the localized content. ChatGPT produced Nepali equivalents for a large set of English strings, including interface actions (e.g., “Sign In” → “साइन इन गर्नुहोस्”), menu items, user prompts, error messages, and array entries like levels and survey-related terms. It also preserved correct grammatical forms, spacing, and punctuation for the Nepali script.  In the final merged code, the vast majority of ChatGPT’s translations were incorporated directly, with only minimal adjustments for style or terminology. Given the near-verbatim adoption of the AI-generated text across hundreds of string entries, the integration rate is estimated at well above 90%, qualifying this as high patch integration.  This case illustrates how ChatGPT can efficiently facilitate large-scale software localization, enabling rapid and consistent translation across an entire application with minimal post-editing required by the developer. |  |
| 89 | <https://github.com/open-learning-exchange/myplanet/pull/2232>  In pull request #2232 of the ole-planet/myplanet repository, the developer added comprehensive Arabic translations for the application’s UI, replacing English text with culturally appropriate and linguistically accurate Arabic equivalents in strings.xml. This work covered menus, action labels, error messages, instructions, and string arrays such as levels, media types, and information categories.  The developer consulted ChatGPT to generate most of these translations, as confirmed by multiple shared conversation links in the PR discussion. ChatGPT provided direct Arabic equivalents for a large set of strings, ensuring proper grammar, punctuation, and terminology. Examples include “Sign In” → “تسجيل الدخول”, “Settings” → “الإعدادات”, and “Beginner” → “مبتدئ”. Even multi-word items and system-specific terms were rendered accurately (e.g., “Flow Video Player” → “مشغل فيديو Flow”, “Manager Login” → “تسجيل الدخول كمدير”).  The final merged code shows that nearly all of ChatGPT’s suggested translations were integrated with minimal adjustments, such as slight stylistic refinements or rewording in a few entries for improved clarity. Given the almost complete adoption across hundreds of entries, the integration rate is estimated above 90%, placing this case in the high patch integration category.  This case demonstrates how ChatGPT can accelerate software localization at scale by producing high-quality translations that require only minor post-editing before production use. | (75-100] |