Common Errors & Examples

Rubrics

[Rubric Error: Misclassifying Critical Rubrics] Critical Rubrics are not included in the rubric list.

For example, if the prompt requested a Python function that adds two numbers, the response must include that function and mention that that function has been added as that would be a Critical Rubric.

Task ID: `6835da563c3f91e41722fee5`

Repo URL: `https://github.com/microsoft/torchgeo`

PR URL (if available): `https://github.com/microsoft/torchgeo/pull/2535`

Commit URL:
`https://github.com/microsoft/torchgeo/commit/42f6dee5ad0ecba61171d7105a8f49477d5a159f`

Cursor question: *How does the SSL4EOL class in @ssl4eo.py dynamically determine which Landsat base class to use for different splits, and how are spectral wavelengths assigned to each band in the dataset samples?*

Critical Rubrics - Response Successes

1. Does the response specify that band wavelengths are retrieved from a class-level dictionary called `wavelengths`?
2. Does the response state that the `self.wavelengths` list is constructed during the class's initialization?

Critical Rubrics - Response Fails

3. Does the response explain that the `self.metadata[split]['all_bands']` list determines the relevant spectral bands?
4. Does the response state that the wavelength tensor is added to the output sample dictionary with the key `wavelength`?
5. Does the response explain that each seasonal timestamp results in one repetition of the band wavelength sequence?

↪ ❌ **Why were these rubrics wrong?** All the existing critical rubrics relate to how the wavelength array is built. No rubric checks how the base class is chosen, which is checked in the code by checking the `split` string.

[Rubric Error: Misclassifying Critical Rubrics] The Contributor marked as "Critical" rubrics that are not critical.

Task ID: `6835da56b72521be80ad46b3`

Repo URL: `https://github.com/oumi-ai/oumi`

PR URL (if available): `https://github.com/oumi-ai/oumi/pull/985`

Commit URL: `https://github.com/oumi-ai/oumi/commit/9131b32cae75d7a7e9046dac987f45e1f9540ff0`

Cursor question: *Can you explain how the mode parameter in functions like `load_pil_image_from_path` and `load_pil_image_from_content_item` within @image_utils.py works in conjunction with the `convert_pil_image_mode` function? Specifically, how does this approach help manage image color modes (e.g., RGB, RGBA) and prevent redundant conversions when loading images from various sources such as file paths, URLs, or binary data?*

Critical Rubrics - Response Fails

3. Does the response provide a specific code example from `src/oumi/utils/image_utils.py` demonstrating how `load_pil_image_from_path` (or a similar function) calls `convert_pil_image_mode internally`?

4. Does the response explain that the `convert_pil_image_mode` function includes a conditional check to compare the current image mode with the requested mode, and returns the original image unmodified if they match, thereby avoiding unnecessary conversion operations?

5. Does the response describe that avoiding unnecessary color mode conversions in image preprocessing such as `skipping image.convert("RGB")` when the image is already in RGB, can reduce image loading latency during batch preprocessing?

↪ ❌ **Why were these rubrics wrong?** The rubric item #3 is referring to a specific implementation that is non-critical because it is not asking for an explanation of how the methods `load_pil_image_from_path` and `load_pil_image_from_content_item` manage image color modes or how they prevent redundant conversions. The rubric item #5 is a good-to-have question that does not respond to the question but provides additional information around it.

[Rubric Error: Misclassifying Critical Rubrics] The Contributor marked as "Critical" rubrics that are not critical.

Task ID: `6835da56e737f34acc23b5d2` **Repo URL:** `https://github.com/jsx-eslint/eslint-plugin-react` **PR URL (if available):** `-` **Commit URL:** `https://github.com/jsx-eslint/eslint-plugin-react/commit/a847c841f6a34a9adde6bc3040a40426a14e9f52`
Cursor question: *How does the filterRules function in eslint-plugin-react inside @all.js process and transform rule configurations, and what are the specific parameters and return values expected when filtering active versus deprecated rules?*
Critical Rubrics - Response Fails
3. Does the response identify at least one predicate commonly used in ESLint configurations, such as filtering rules with `meta.docs.recommended`? *4. Does the response mention that `filterRules` does not include any TypeScript type annotations in its declaration because it is written in plain JavaScript?* *5. Does the response state that `filterRules` does not mutate the original `rules` object, because it instead returns a new filtered object?*
Reference Answer:

I'll help you understand how the `filterRules` function works in eslint-plugin-react. Let me search for this function and analyze its implementation.
Search files...
Perfect! I found the `filterRules` function in the `configs/all.js` file. Let me read the complete file to understand the full context and implementation.
Read file: configs/all.js
Now let me also check what the `allRules` object looks like to better understand the structure of the rules being filtered.
Read file: lib/rules/index.js
Let me also check an example of a rule to understand the structure of the `meta.deprecated` property that the `filterRules` function is checking.
Read file: lib/rules/jsx-space-before-closing.js
Now let me check a non-deprecated rule to see the contrast:
Read file: lib/rules/jsx-key.js
Perfect! Now I have a comprehensive understanding of how the `filterRules` function works. Let me provide a detailed explanation.

How the `filterRules` function works in eslint-plugin-react

The `filterRules` function in `configs/all.js` is a utility function that processes and transforms rule configurations by filtering them based on specific criteria. Here's a detailed breakdown:

Function Signature and Parameters

```8:10:configs/all.js
function filterRules(rules, predicate) {
  return fromEntries(entries(rules).filter((entry) => predicate(entry[1])));
}
```

Parameters:
- `rules` (Object): An object containing all ESLint rules, where each key is a rule name and each value is the rule definition object
- `predicate` (Function): A callback function that receives a rule definition and returns a boolean to determine if the rule should be included in the filtered result

Return Value:
- Returns a new object containing only the rules that pass the predicate test

Processing Logic

The function works through these steps:

1. **Convert to entries**: Uses `entries(rules)` to convert the rules object into an array of `[key, value]` pairs
2. **Filter entries**: Applies the `filter()` method with the predicate function, where `entry[1]` represents the rule definition (the value part of each key-value pair)
3. **Reconstruct object**: Uses `fromEntries()` to convert the filtered array back into an object

Specific Usage in eslint-plugin-react

The function is used twice in the configuration to separate active and deprecated rules:

Filtering Active Rules
```16:16:configs/all.js
const activeRules = filterRules(allRules, (rule) => !rule.meta.deprecated);
```

- **Predicate**: `(rule) => !rule.meta.deprecated`
- **Purpose**: Selects rules where `rule.meta.deprecated` is falsy (undefined, false, etc.)
- **Result**: Contains only non-deprecated, active rules

Filtering Deprecated Rules
```19:19:configs/all.js
const deprecatedRules = filterRules(allRules, (rule) => rule.meta.deprecated);
```

- **Predicate**: `(rule) => rule.meta.deprecated`
- **Purpose**: Selects rules where `rule.meta.deprecated` is truthy
- **Result**: Contains only deprecated rules

Rule Structure Context

Each rule in the `allRules` object has this structure:
- **Active rule example** (like `jsx-key`): Has `meta` object without `deprecated` property
- **Deprecated rule example** (like `jsx-space-before-closing`): Has `meta.deprecated: true` and often includes `meta.replacedBy` array

Expected Parameters and Return Values

When filtering active rules:
- **Input**: Complete rules object from `lib/rules/index.js`
- **Predicate**: Function that returns `true` for non-deprecated rules
- **Output**: Object containing only active rules (those without `meta.deprecated: true`)

When filtering deprecated rules:
- **Input**: Same complete rules object
- **Predicate**: Function that returns `true` for deprecated rules
- **Output**: Object containing only deprecated rules (those with `meta.deprecated: true`)

The filtered results are then used to:
- Configure active rules as errors in the ESLint configuration
- Expose deprecated rules separately for reference
- Maintain backward compatibility while clearly separating rule states

This approach allows the plugin to maintain a clean separation between current and legacy rules while providing users with comprehensive rule sets based on their needs.

[Not Being Self-Contained] Each rubric item must contain all the information within the rubric that is needed to determine whether the response is True or False.

❌ ***Bad Examples***	✅ ***Good Examples of Self Containment***
1. Does the response explain what an EC2 PlacementGroup is? *2. Does the response describe the purpose of the `Strategy` property in a `PlacementGroup`?* *3. Does the response describe the purpose of the `SpreadLevel` property in a `PlacementGroup`?* *4. Does the response explain the importance of validating these properties early to prevent template deployment failures?* *5. Does the response explain the creation flow of a `PlacementGroup`?* *6. Does the response explain at what point in the creation of a `PlacementGroup` the validation function for `Strategy` is applied?* *7. Does the response explain at what point in the creation of a `PlacementGroup` the validation function for `SpreadLevel` is applied?* ↪ **Why was this rubric wrong?** These rubrics require searching for external information to be answered: **item #1** can be improved by explaining what an EC2 placement is, and avoiding leaving an open question when asking the "what". **Item #2** doesn't describe the purpose of the `Strategy`property, but should. **Item #3** can be improved by providing what makes qualified lateral functions useful. **Item #4** writes "these properties," but which properties are being referred to is not included in the question. **Item #5** doesn't specify what the creation flow is. **Item #6** doesn't mention at which point the validation function `Strategy` is applied. Same issue with **Item #7** with `SpreadLevel`	✅ *1. Does the response explain that EC2 PlacementGroup is a logical grouping strategy that controls the physical arrangement of Amazon EC2 instances on the underlying hardware infrastructure?* ✅ *2. Does the response describe the purpose of the `Strategy` property in a `PlacementGroup` as specifying the resource distribution method with allowed values, such as "cluster", "partition", or "spread"?* ✅ *3. Does the response describe the purpose of the `SpreadLevel` property in a `PlacementGroup` as determining the distribution granularity for resources, allowing specification at either "host" or "rack" level when using the "spread" placement strategy?* ✅ *4. Does the response explain that the importance of validating both the `Strategy` and `SpreadLevel` properties early is to prevent invalid configurations from propagating through the system?* ✅ *5. Does the response explain that the importance of validating both the `Strategy` and `SpreadLevel` properties early is to provide immediate feedback about configuration errors?* ✅ *6. Does the response explain that the creation flow of a `PlacementGroup` requires specifying a `Strategy` first and then a `SpreadLevel`?* ✅ *7. Does the response explain that the validation function for `Strategy` in a `PlacementGroup` is applied when a user configures or creates a `PlacementGroup` with spread distribution strategy in an AWS CloudFormation template using Troposphere?* ✅ *8. Does the response explain that the validation function for `SpreadLevel` in a `PlacementGroup` is applied when a user configures or creates a `PlacementGroup` with spread distribution strategy in an AWS CloudFormation template using Troposphere?*

[Error: Overlapping] Having overlapping criteria

Each rubric item should deal with one specific dimension. This is because the same error from a model shouldn't be punished multiple times.

❌ ***Bad Examples***	✅ ***Good Examples***
1. Does the code correctly implement the `convert_parameter_to_seconds` decorator to ensure the `duration` parameter is always in seconds? *2. Does the code introduce the `_mono_factor_getter` function to compute the fadeout factor for mono audio clips?* *3. Does the code introduce the `_stereo_factor_getter` function to compute the fadeout factor for stereo audio clips, ensuring proper handling of multiple audio channels?* *4. Does the code refactor the fading logic into the helper functions `_mono_factor_getter` and `_stereo_factor_getter` without introducing regressions?* *5. Does the code replace the inline fading logic in `audio_fadeout` with a call to the appropriate helper function based on the number of audio channels (`clip.nchannels`)?* *6. Does the code ensure that the `audio_fadeout` function still works correctly for both mono and stereo audio clips after the changes?* *7. Does the code modify the `audio_fadeout` function to include docstring with sections for `Parameters` and `Examples` explaining their usage?* *8. Does the updated docstring for `audio_fadeout` accurately describe the function's behavior, the `duration` parameter, and provide a valid usage example?*	*1. Does the code implement the decorator to ensure the duration is always in seconds?* *2. Does the code introduce a getter function to compute the fadeout factor for mono audio clips?* *3. Does the code introduce a getterfunction to compute the fadeout factor for stereo audio clips, ensuring proper handling of multiple audio channels?* *4. Does the code refactor the fading logic into the helper functions without introducing regressions?* *5. Does the code replace the inline fading logic with a call to the appropriate helper function based on the number of audio channels?* *6. Does the code ensure that the audio fadeout functionality still works for the audio clips after the changes?* *7. Does the code include docstrings with sections for the expected behavior of the audio fadeout functionality?*

↪ **Why was this rubric wrong?** The code rubric items #7 and #8 here are referring to documenting the function `audio_faceout` with docstrings, and both refer, for example, to adding an example in the docstrings.

[Not Being Self-Contained] Each rubric item must contain all the information within the rubric that is needed to determine whether the response is True or False.

❌ ***Bad Examples***	✅ ***Good Examples***
1. Does the response describe how the `LATERAL_FUNCTION_AS_VIEW` flag allows functions to be treated as table expressions in `sqlglot/dialects/postgres.py`? *2. Does the response explain that using `"AS {columns}" instead of "({columns})"` changes the column formatting when handling lateral view syntax in `sqlglot/generator.py`?* *3. Does the response explain that lateral functions allow subqueries in the `FROM` clause to reference columns from preceding items in the `FROM` clause?* *4. Does the response provide examples of the supported syntax?* *5. Does the response explain that a qualified function is a function that is associated with specific database objects or schemas, often referred to by their fully qualified name (e.g., schema.function_name)?* *6. Does the response provide examples of why qualified lateral functions are useful?*	*1. Does the response describe that the `LATERAL_FUNCTION_AS_VIEW` flag allows functions to be treated as table expressions?* *2. Does the response explain that using `"AS {columns}" instead of "({columns})"` changes the column formatting when handling lateral view syntax in `sqlglot/generator.py`?* *3. Does the response explain that lateral functions allow subqueries in the `FROM` clause to reference columns from preceding items in the `FROM` clause?* *4. Does the response provide examples of the supported syntax of qualified function names with dot notation for the new code?* *5. Does the response explain that a qualified function is a function that is associated with specific database objects or schemas, often referred to by their fully qualified name (e.g., schema.function_name)?* *6. Does the response provide examples of why qualified lateral functions are useful such as for working with semi-structured data like JSON or arrays, creating sophisticated joins with dependencies, or performing row-by-row transformations with custom functions?*

↪ **Why was this rubric wrong?** The code rubrics required searching for external information to be answered: item #1 can be improved by explaining what the expected behavior is, and avoiding leaving an open question when asking the “how”, and item #6 can be improved by providing what makes qualified lateral functions useful.

[Vague Descriptions] Using general and undescriptive terms like "appropriate", "correctly", “good”, etc., without specifying what is appropriate/good.

❌ ***Bad Examples***	✅ ***Good Examples***
❌ E.g., *“Does the response include the accurate default parameters?"* is a vague description.	✅ *“Does the response include a parsing function that takes a string as parameter”*
❌ E.g., *“Does the response provide the appropriate reasoning for why empty DataFrames are treated differently and excluded from the warning logic?”*	✅ *“Does the response explain that empty DataFrames are treated differently and excluded from the warning logic because…?”*
❌ *Does the response provide good examples of why qualified functions are useful?*	✅ *Does the response provide examples of the usefulness of qualified lateral functions, such as for working with semi-structured data like JSON or arrays, creating sophisticated joins with dependencies, or performing row-by-row transformations with custom functions?*

↪ **Why were these rubrics wrong?** They use words that do not reference any premise in particular. It is not possible to objectively evaluate these items as true or false.

[Not Being Atomic] Bundling multiple criteria into a single rubric: Each rubric question should evaluate exactly one distinct aspect.

❌ ***Bad Examples***	✅ ***Good Examples***
1. Does the response generate `PARTITION BY RANGE` and `UNIQUE KEY` constructs according to StarRocks syntax? *2. Does the response add new expression classes in `sqlglot/expressions.py`?* *3. Does the response integrate new expression classes into the parsing and generation system?*	*1. Does the response generate `PARTITION BY RANGE` constructs according to StarRocks syntax?* *2. Does the response generate `UNIQUE KEY` constructs according to StarRocks syntax?* *3. Does the response add new expression classes?* *4. Does the response integrate new expression classes into the parsing and generation system?*

4. Does the response update the tokenizer, parser, generator, and expression modules modularly without disrupting unrelated functionality? *5. Does the response map the `LARGEINT` type in the type mapping of the StarRocks dialect?* *6. Does the response update `PROPERTY_PARSERS` and `PROPERTIES_LOCATION` to recognize and place the new properties?* *7. Does the response ensure that parsing composite keys supports duplicate and unique keys through a shared parsing helper?* *8. Does the response include the SQL generation methods in `sqlglot/generator.py` for `PARTITION BY RANGE` and `UNIQUE KEY`?* *9. Does the response parse dynamic range partitioning into an object with `start`, `end`, and `every` attributes?* *10. Does the response generate valid SQL strings for `UNIQUE KEY (...)`, `PARTITION BY RANGE (...)`, and dynamic ranges using `START ... END ... EVERY ...`?*	*5. Does the response update the tokenizer module modularly without disrupting unrelated functionality?* *6. Does the response update the parser module modularly without disrupting unrelated functionality?* *7. Does the response update the generator module modularly without disrupting unrelated functionality?* *8. Does the response update the expression module modularly without disrupting unrelated functionality?* *9. Does the response map the `LARGEINT` type in the type mapping of the StarRocks dialect?* *10. Does the response update `PROPERTY_PARSERS` to recognize the new properties?* *11. Does the response update `PROPERTIES_LOCATION` to place the new properties?* *12. Does the response ensure that parsing composite keys supports duplicate and unique keys through a shared parsing helper?* *13. Does the response include the SQL generation methods in `sqlglot/generator.py` for `PARTITION BY RANGE` that generate valid SQL strings?* *14. Does the response include the SQL generation methods in `sqlglot/generator.py` for `UNIQUE KEY` that generate valid SQL strings?* *15. Does the response generate valid SQL for dynamic partitions using `START ... END ... EVERY ...`?*

Prompt

Common Errors in the Questions	**You Should:**
❌ **Not being natural enough:** Your prompts need to be realistic and be something a Software Developer may naturally ask in Cursor. ❌ Don't ask several questions into one just to make the prompt more complex. While it might be natural to have a follow up question to your main one, prompts with more than 2 questions are not natural. ❌ **Copying the Issue Description in the prompt:** The issue description should be used for understanding and for getting inspiration, but the prompt shouldn't be too similar to it as it wouldn't create a significantly new datapoint. • If the issue description has replication code, don't just copy it. You can change the variable names, for example, to adapt it to another use case. ❌ **Being too prescriptive:** While the prompt needs to provide enough context and guidance, it shouldn't tell the model exactly what to do with the exact implementation from the PR. ❌ **Being too vague:** The question should be realistic and have a specific answer to it. Don't just write extremely broad and open-ended questions. ❌ **Not providing the file context in the prompt using @.** ❌ **Writing prompts that look like GitHub issue descriptions:** Think about what a developer might ask a coding agent when debugging code. • Avoid using fancy formatting: Avoid excessive use of backticks and markdown. Bullet points are fine, but try to avoid them to make the prompt more natural. ❌ **DO NOT mention the PR or the commit in the prompt!**	✅ **Use a natural and casual tone:** Don't be formal! Remember you are a software developer talking to an AI agent. ✅ **Focus on writing one question**. You can add constraints and requirements to make it more specific. ✅ **Provide context:** Include enough context about the problem and instructions on what the LLM should achieve, so that it has a good idea of what to address. • You can do this by stating the relevant existing code or behavior: The prompt should have the relevant contextual information to guide the model. • You can use @file_name to reference file names in Cursor. ✅ **Reference relevant files:** Reference other files by using the @file_name convention in Cursor that provides that file as context. ✅ **Be precise:** Try to match the level of precision you'd normally provide when prompting an LLM. Think about how you normally use ChatGPT or other models when coding, and write prompts that match your "best" prompts to ChatGPT. ✅ **Be careful with the level of detail,** try to avoid being too prescriptive when writing the prompt. • This means that if your prompt deals with code changes, you shouldn't just give the agent a list with all the specific code changes that are given in the PR. Instead, try to prompt the LLM in a way that gives room for different solutions, not just the one in the Gold Patch ✅ **Explain any constraints or requirements.**