Automatically Updating Deprecated API Usage through LLMs and Documentation Retrieval

Sarah Nadi & Max Schaefer

Coding with GitHub Copilot

My thoughts and experience on the new GitHub Copilot tool.

Jul 14, 2021 • 12 min read

programming

Usage of outdated APIs

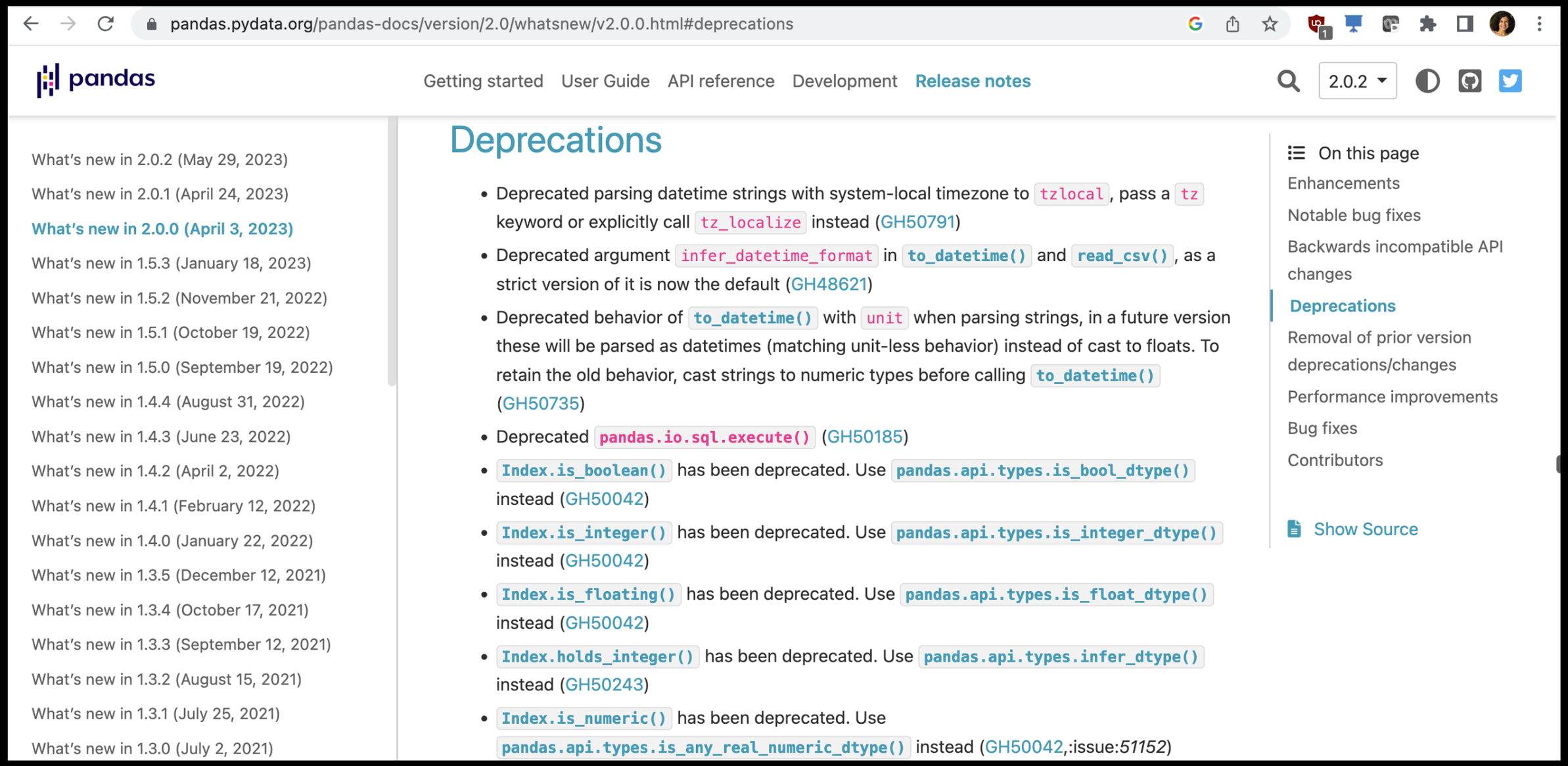
As an ML researcher and developer, I am typically working with the latest ML frameworks and tools. However, GitHub Copilot is trained on older codebases and does not have knowledge of these cutting-edge tools and is often unable to provide relevant suggestions.

I first discovered this issue when trying to write fastai-related code and get GitHub Copilot to provide relevant suggestions. However, since the latest version of fastai was only released in August 2020, GitHub Copilot was not able to provide any relevant suggestions and instead provided code for using older versions of fastai. This indicates that the codebases that GitHub Copilot is trained on must be at least before August 2020, if not earlier. Similarly, I discovered that GitHub Copilot was unable to provide any suggestions regarding the usage of the timm library, which is one of the leading deep learning+computer vision libraries.

Goal

Automatically update code that uses removed or deprecated APIs

Insights



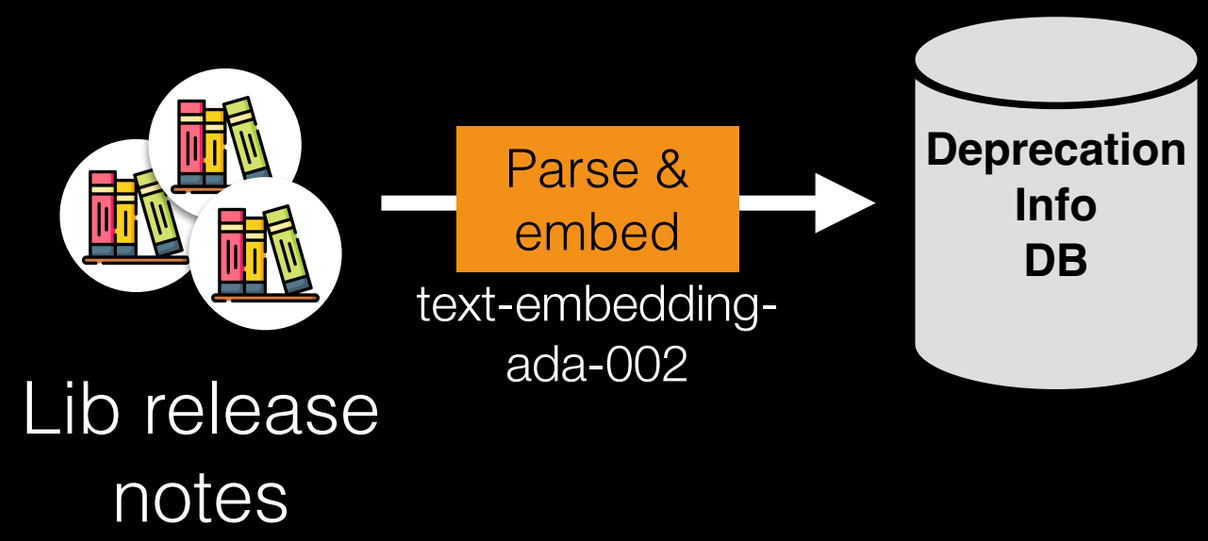
Release notes often contain info about deprecations and their alternatives/replacements

Insights

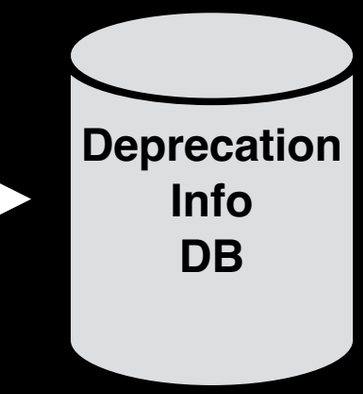
Release notes often contain info about deprecations and their alternatives/replacements

Large Language Models (LLMs) may be able to “understand” these descriptions and translate them to code changes

Proposed Technique



Proposed Technique

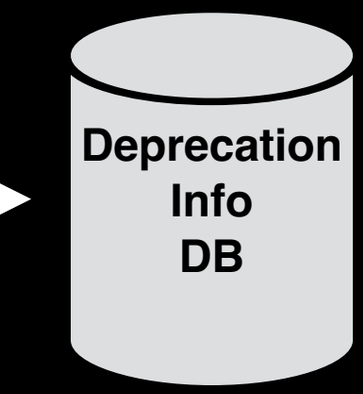


Proposed Technique

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.is_mixed():
    print('mixed type')
```

Newly added code

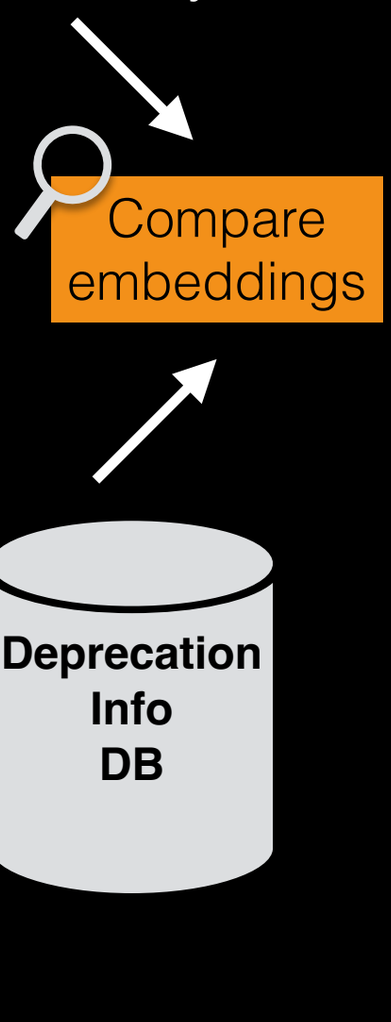


Proposed Technique

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.is_mixed():
    print('mixed type')
```

Newly added code

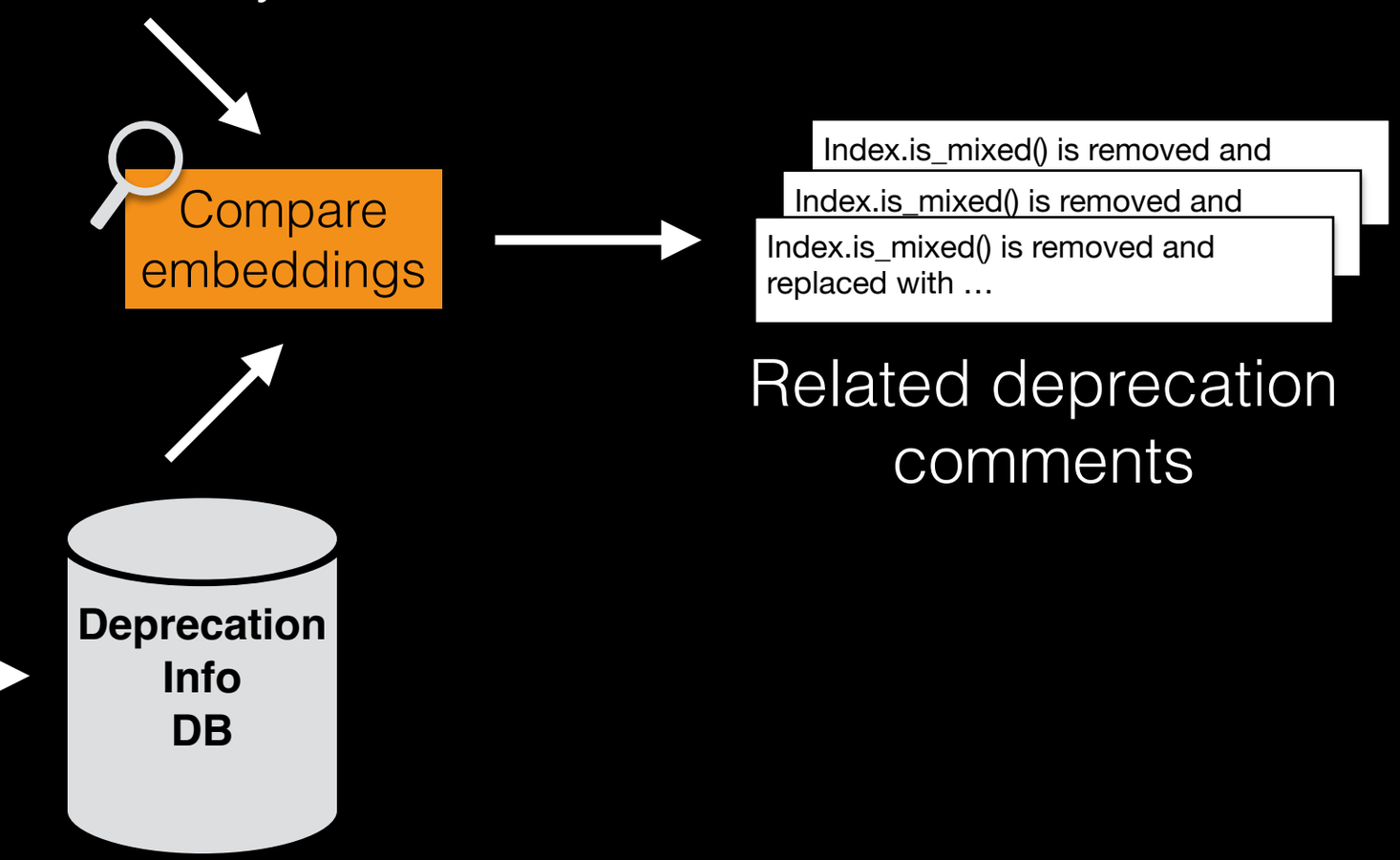


Proposed Technique

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.is_mixed():
    print('mixed type')
```

Newly added code



Proposed Technique

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.is_mixed():
    print('mixed type')
```

Newly added code

Compare embeddings

Deprecation Info DB

Index.is_mixed() is removed and

Index.is_mixed() is removed and

Index.is_mixed() is removed and replaced with …

Related deprecation comments

gpt-turbo3.5/gpt4

Proposed Technique

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.is_mixed():
    print('mixed type')
```

Newly added code

Compare embeddings

Deprecation Info DB

Index.is_mixed() is removed and

Index.is_mixed() is removed and

Index.is_mixed() is removed and replaced with …

Related deprecation comments

gpt-turbo3.5/gpt4

```
import pandas

idx = pandas.Index([0,'1',3, 'fooo'])
if idx.inferred_type == 'mixed':
    print('mixed type')
```

Updated code w/ explanation

Prompt in the Background

````
Given the provided numbered reference information, decide if the provided code needs to be updated.
Focus only on updates that do not change the code's functionality and are related to outdated, deprecated, or non-existent APIs.
You must reply in the following exact numbered format.
1. ```The full updated code snippet in a fenced code block``` or an empty fenced code block if you don't want to update the code
2. Reason for update (if any)
3. List of reference numbers used (if any) to update the code. If none of the references below were useful, say 'No references used'

Provided code:

```
$original_code
```

Provided reference information:

$references

Your Response:
````

Example Prompt

````
Given the provided numbered reference information, decide if the provided code needs to be updated.
Focus only on updates that do not change the code's functionality and are related to outdated, deprecated, or non-existent APIs.
You must reply in the following exact numbered format.
1. ```The full updated code snippet in a fenced code block``` or an empty fenced code block if you don't want to update the code
2. Reason for update (if any)
3. List of reference numbers used (if any) to update the code. If none of the references below were useful, say 'No references used'

Provided code:

```
import numpy as np

myArray = np.array([[0.434, 0.768, 0.54900530],
                    [0.36211, 0.3784, 0.2415],
                    [0.258, 0.52929049, 0.39172155]])

sorted = np.msort(myArray)

print(f"Min element is {sorted[0][0]}")


```

Provided reference information:
````

Main takeaway:
Model receives a (long) list of references

```
1. The numpy.msort function is deprecated. Use np.sort(a, axis=0) instead.
2. The mini() method of np.ma.MaskedArray has been removed. Use either np.ma.MaskedArray.min() or np.ma.minimum.reduce().
3. The mini() method of np.ma.MaskedArray has been removed. Use either np.ma.MaskedArray.min() or np.ma.minimum.reduce().
4. The single-argument form of np.ma.minimum and np.ma.maximum has been removed. Use np.ma.minimum.reduce() or np.ma.maximum.reduce() instead.
5. The single-argument form of np.ma.minimum and np.ma.maximum has been removed. Use np.ma.minimum.reduce() or np.ma.maximum.reduce() instead. (gh-22228)
6. Removed the alias scipy.random for numpy.random.
7. For example:  In [3]: ser[2:4] Out[3]: 5    3 7    4 dtype: int64
8. The scalar type aliases ending in a 0 bit size: np.object0, np.str0, np.bytes0, np.void0, np.int0, np.uint0 as well as np.bool8 are now deprecated and will eventually be removed.
9. Many of these did succeed before.  Such code was mainly useful for unsigned integers with negative values such as np.uint8(-1) giving np.iinfo(np.uint8).max.
10. Bug in DataFrame.pivot_table with sort=False results in sorted index (17041)
11. Note that conversion between NumPy integers is unaffected, so that np.array(-1).astype(np.uint8) continues to work and use C integer overflow logic.  For negative values, it will also work to view the array: np.array(-1,
dtype=np.int8).view(np.uint8). In some cases, using np.iinfo(np.uint8).max or val % 2**8 may also work well.
12. Old behavior:  In [3]: ser[2:4] Out[3]: 5    3 7    4 dtype: int64
13. Multidimensional indexing with non-tuple values is not allowed.  Previously, code such as arr[ind] where ind = [[0, 1], [0, 1]] produced a FutureWarning and was interpreted as a multidimensional index (i.e., arr[tuple(ind)]).
Now this example is treated like an array index over a single dimension (arr[array(ind)]).  Multidimensional indexing with anything but a tuple was deprecated in NumPy 1.15.
14. Most of the time setting values with DataFrame.iloc attempts to set values inplace, only falling back to inserting a new array if necessary. There are some cases where this rule is not followed, for example when setting an
entire column from an array with different dtype:  In [3]: df.iloc[:, 0] = new_prices In [4]: df.iloc[:, 0] Out[4]: book1    98 book2    99 Name: price, dtype: int64 In [5]: original_prices Out[5]: book1    11.1 book2    12.2
Name: price, float: 64
15. Future behavior:  In [4]: ser.loc[2:4] Out[4]: 2    1 3    2 dtype: int64
16. The numpy.fastCopyAndTranspose function has been deprecated. Use the corresponding copy and transpose methods directly:  arr.T.copy()
17. Changed behavior of Index.ravel to return a view on the original Index instead of a np.ndarray (36900)
18. scipy.spatial.distance now enforces that the input vectors are one-dimensional.
19. Attempting a conversion from a Python integer to a NumPy value will now always check whether the result can be represented by NumPy.  This means the following examples will fail in the future and give a DeprecationWarning
now:  np.uint8(-1) np.array([3000], dtype=np.int8)
20. Series.unique with dtype "timedelta64[ns]" or "datetime64[ns]" now returns TimedeltaArray or DatetimeArray instead of numpy.ndarray (49176)
21. Series.unique with dtype "timedelta64[ns]" or "datetime64[ns]" now returns TimedeltaArray or DatetimeArray instead of numpy.ndarray (49176)

Your Response:
```

Does this work?

Experiment Setup

Experiment Setup

- 13 examples from 4 different Python libraries (pandas, numpty, scipy, networx)

Experiment Setup

- 13 examples from 4 different Python libraries (pandas, numpty, scipy, networx)
- Populate the DB with deprecation info from all release notes since May 2022 for all 4 libs

Experiment Setup

- 13 examples from 4 different Python libraries (pandas, numpty, scipy, networx)
- Populate the DB with deprecation info from all release notes since May 2022 for all 4 libs
- Evaluate the updated code (is deprecated API removed?)

Experiment Setup

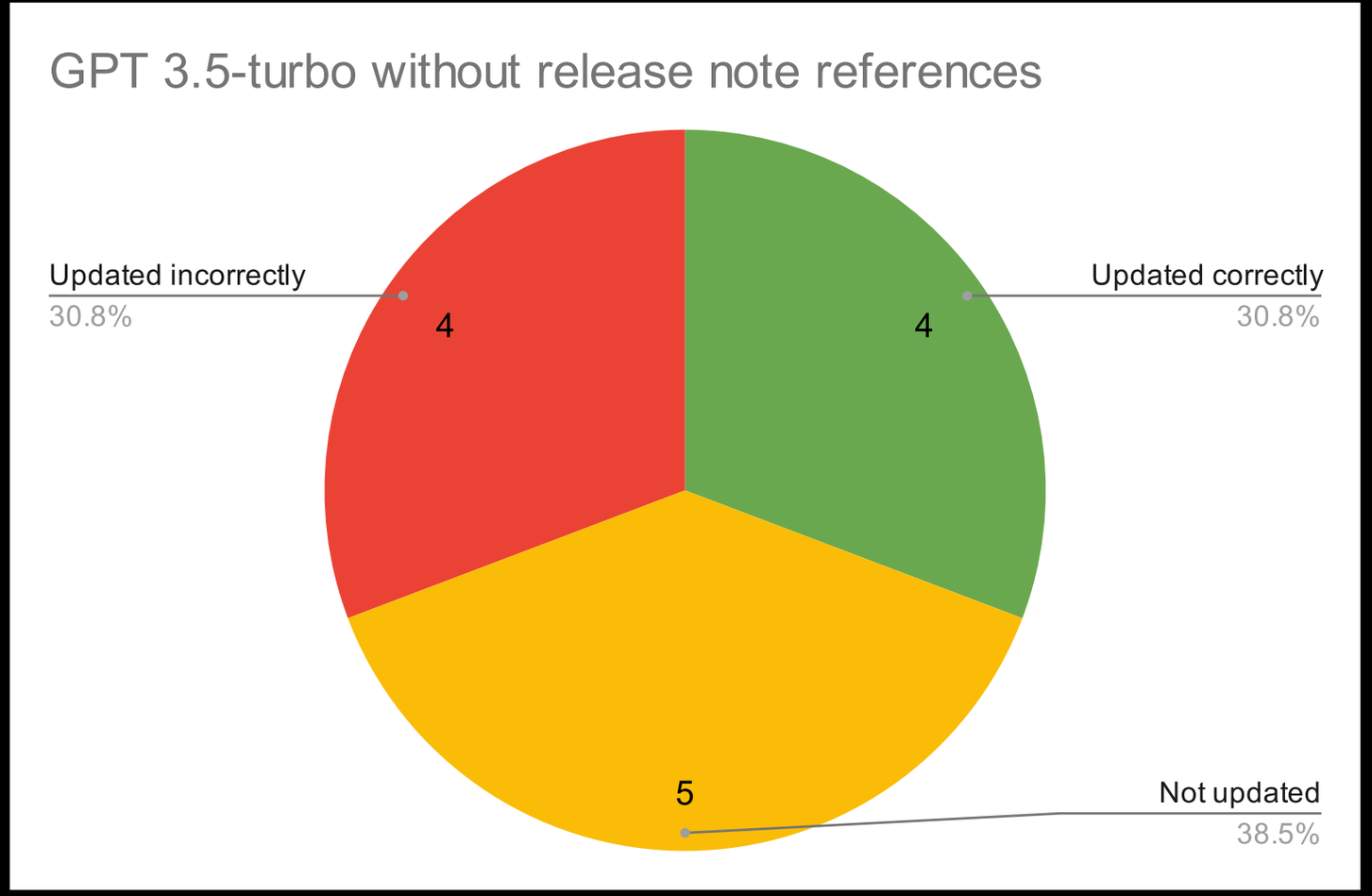
- 13 examples from 4 different Python libraries (pandas, numpty, scipy, networx)
- Populate the DB with deprecation info from all release notes since May 2022 for all 4 libs
- Evaluate the updated code (is deprecated API removed?)
    - Compare model only to model+doc retrieval (does retrieval help?)

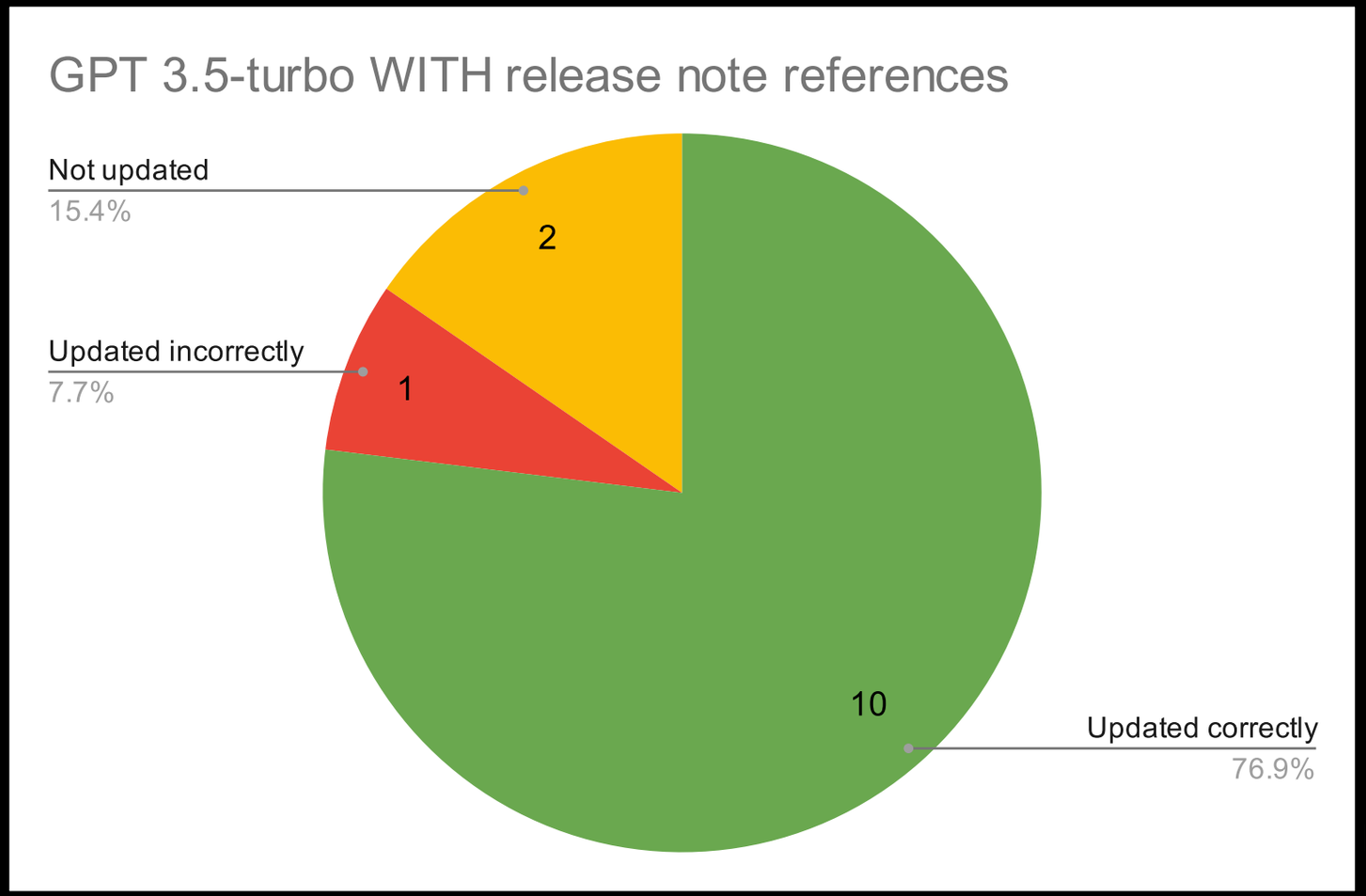
Experiment Setup

- 13 examples from 4 different Python libraries (pandas, numpty, scipy, networx)
- Populate the DB with deprecation info from all release notes since May 2022 for all 4 libs
- Evaluate the updated code (is deprecated API removed?)
    - Compare model only to model+doc retrieval (does retrieval help?)
    - Compare gpt3.5-turbo to gpt4 (is gpt4 better?)

Results

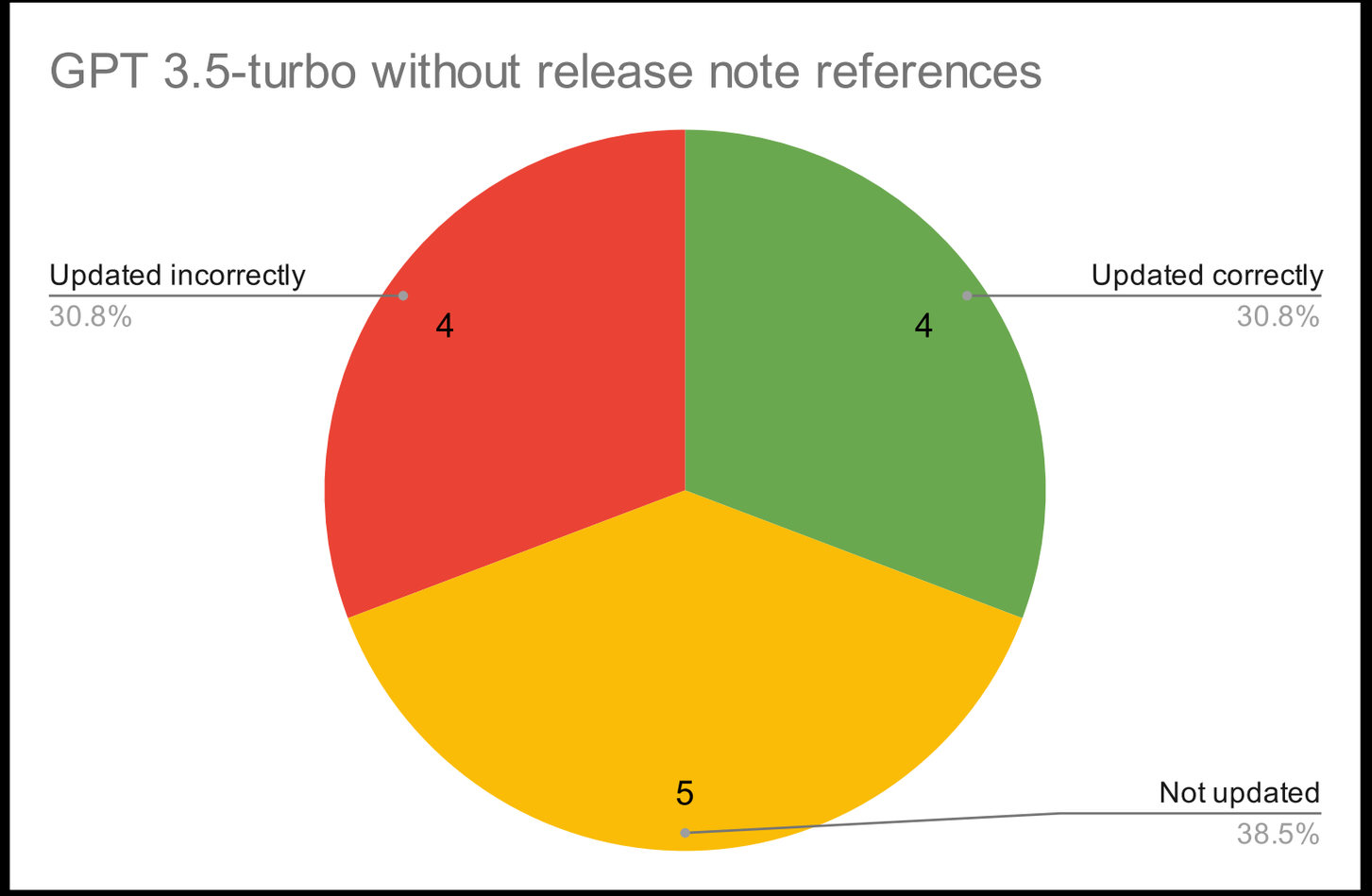
GPT 3.5-turbo

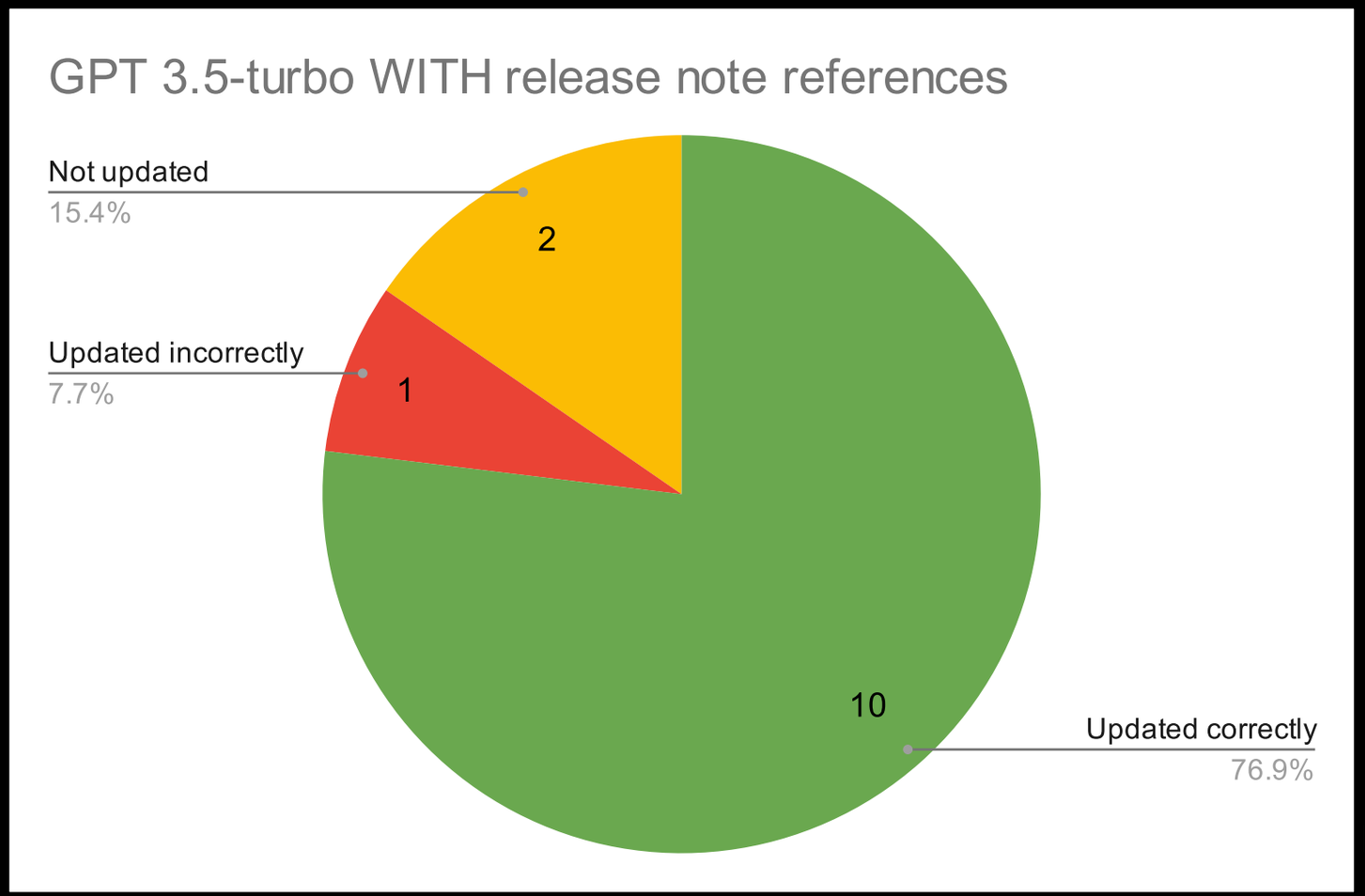

GPT 3.5-turbo without release note references
Updated incorrectly
30.8%
4
4
Updated correctly
30.8%
5
Not updated
38.5%


GPT 3.5-turbo WITH release note references
Not updated
15.4%
2
Updated incorrectly
7.7%
1
10
Updated correctly
76.9%

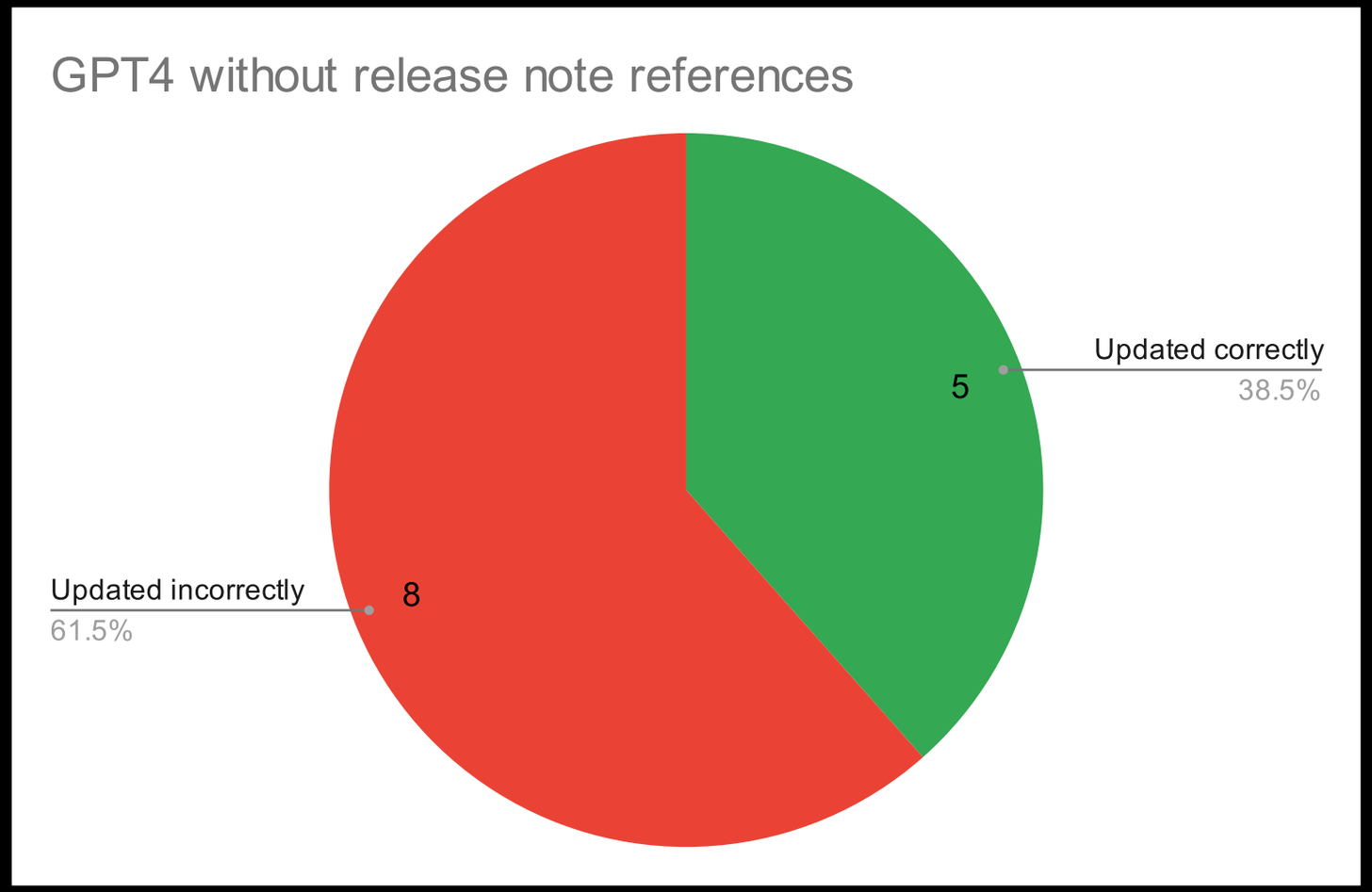
GPT 3.5-turbo

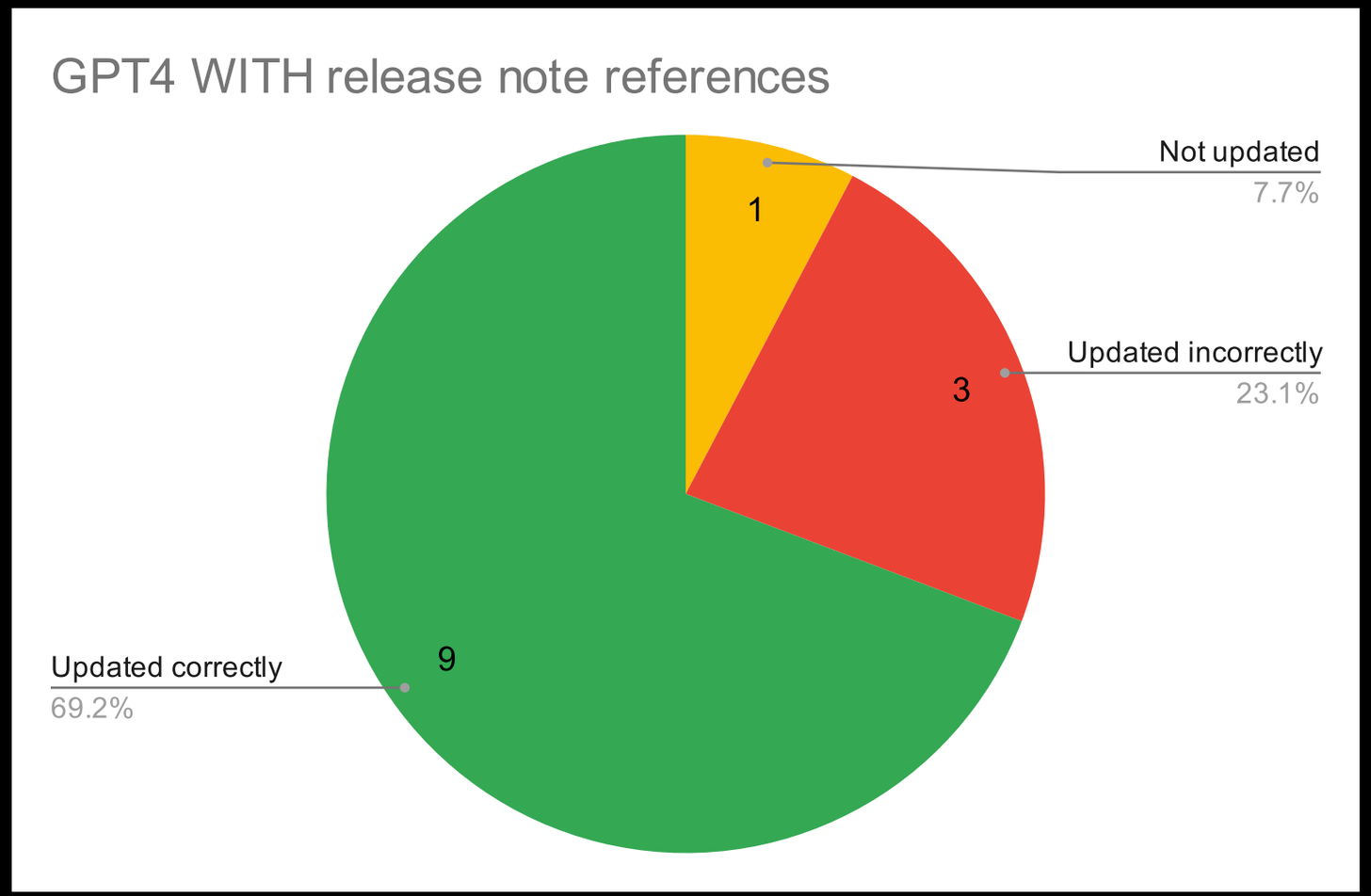
Providing the model with info from release notes allows it to update more examples correctly and reduce incorrect updates





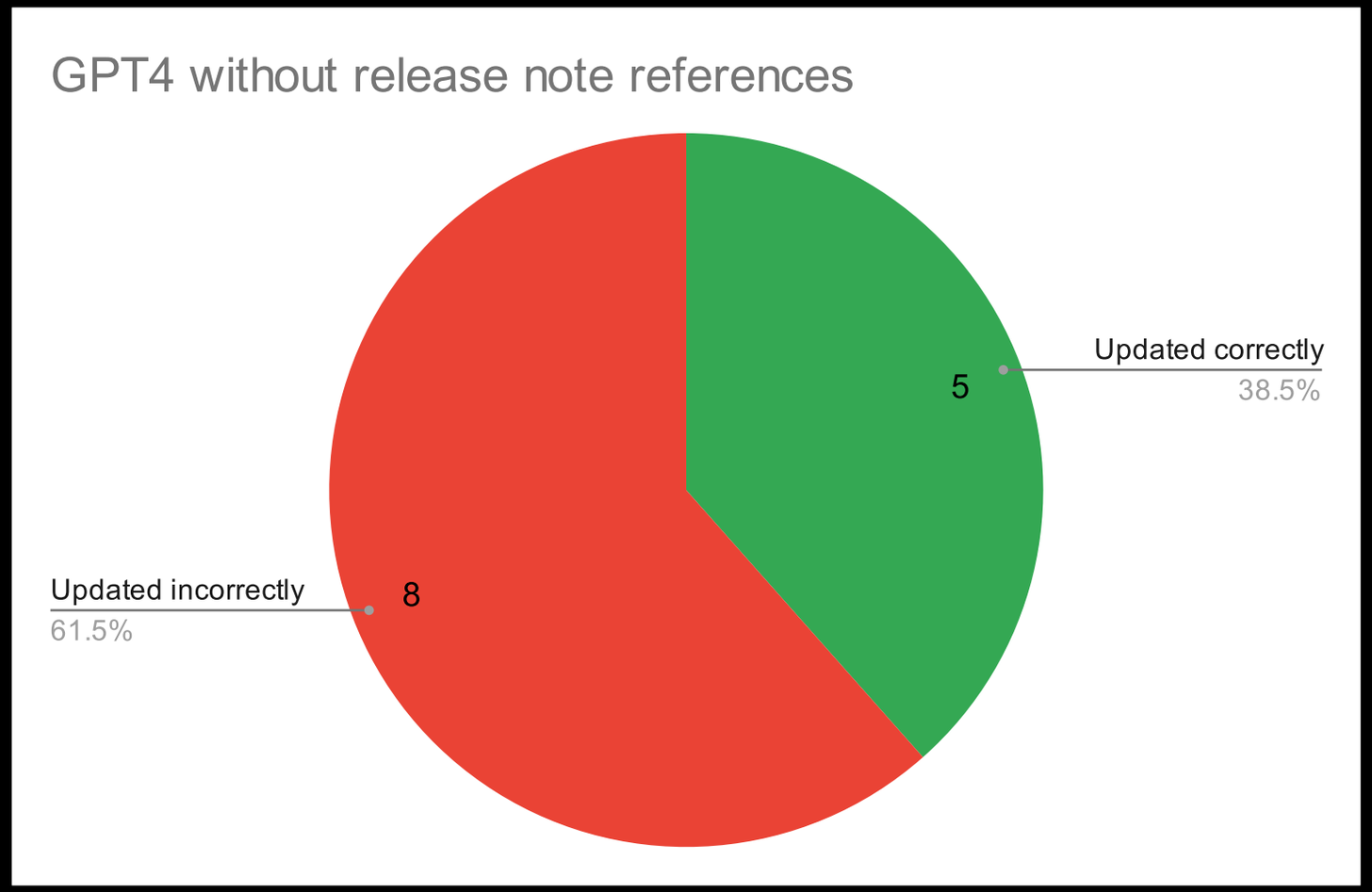
GPT 4

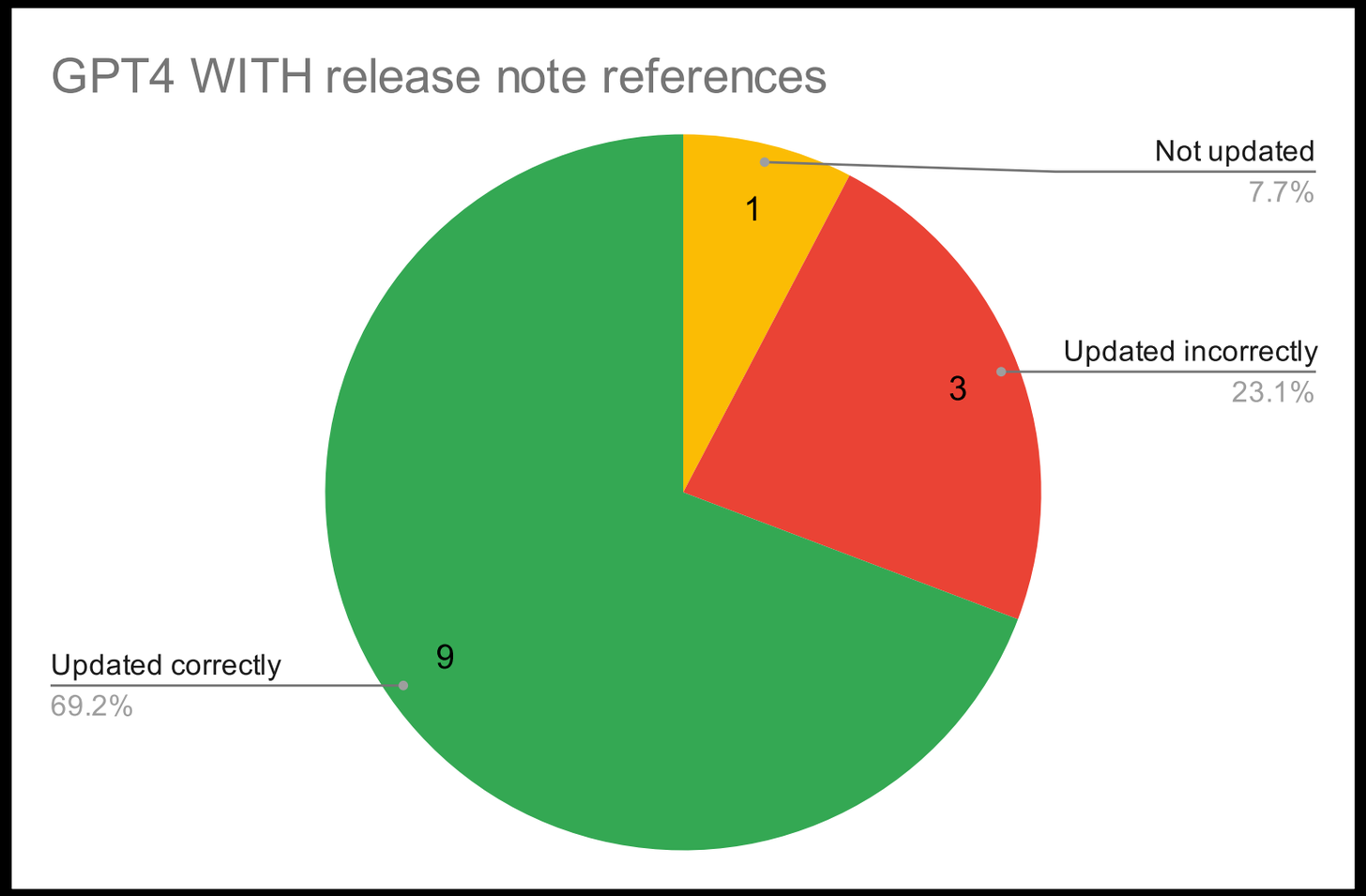

GPT4 without release note references
Updated correctly
38.5%
5
Updated incorrectly
61.5%
8


GPT4 WITH release note references
Not updated
7.7%
1
Updated incorrectly
23.1%
3
Updated correctly
69.2%
9

GPT 4

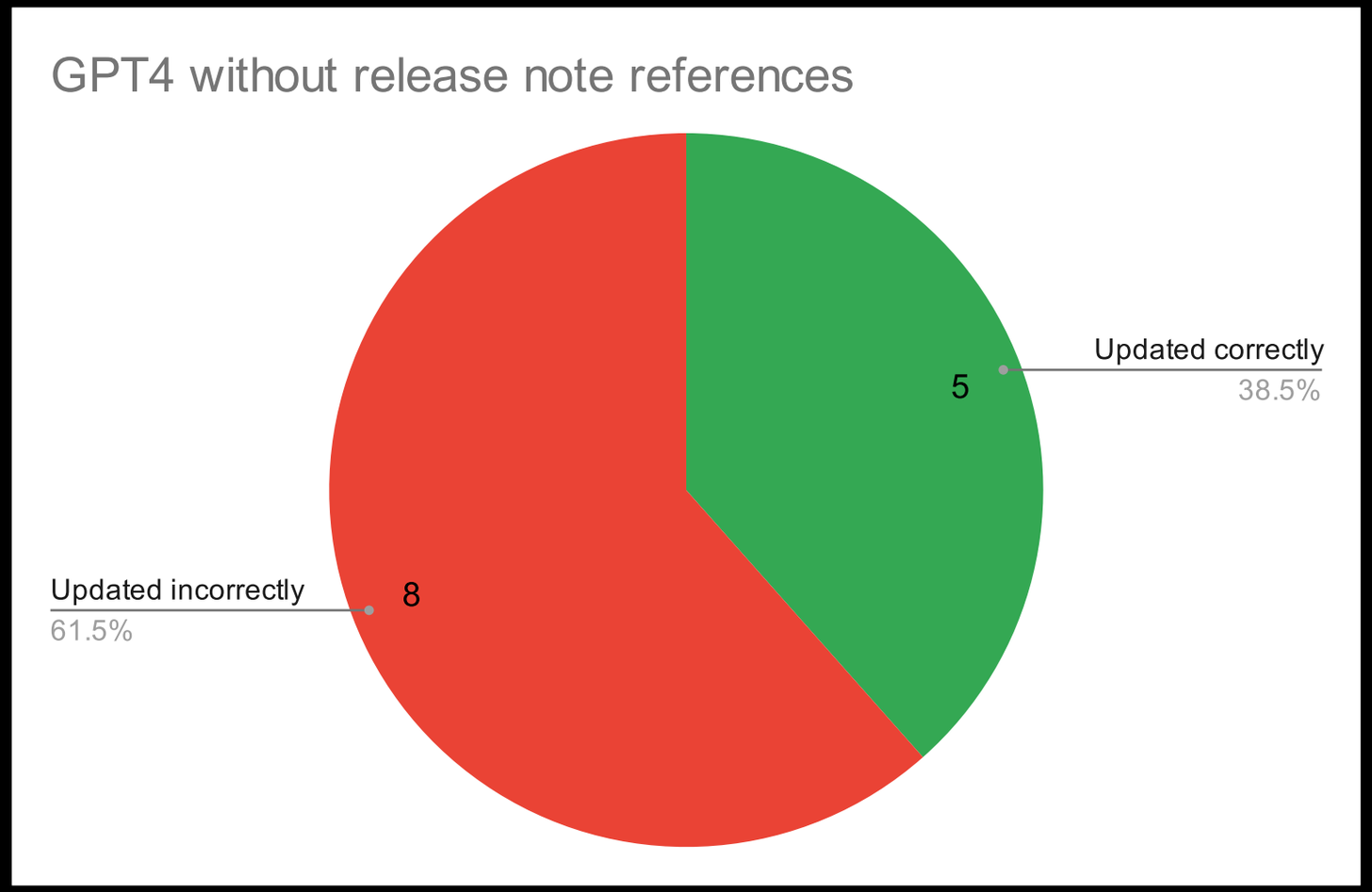
GPT4 alone always tries to update the code, resulting in lots of incorrect updates.



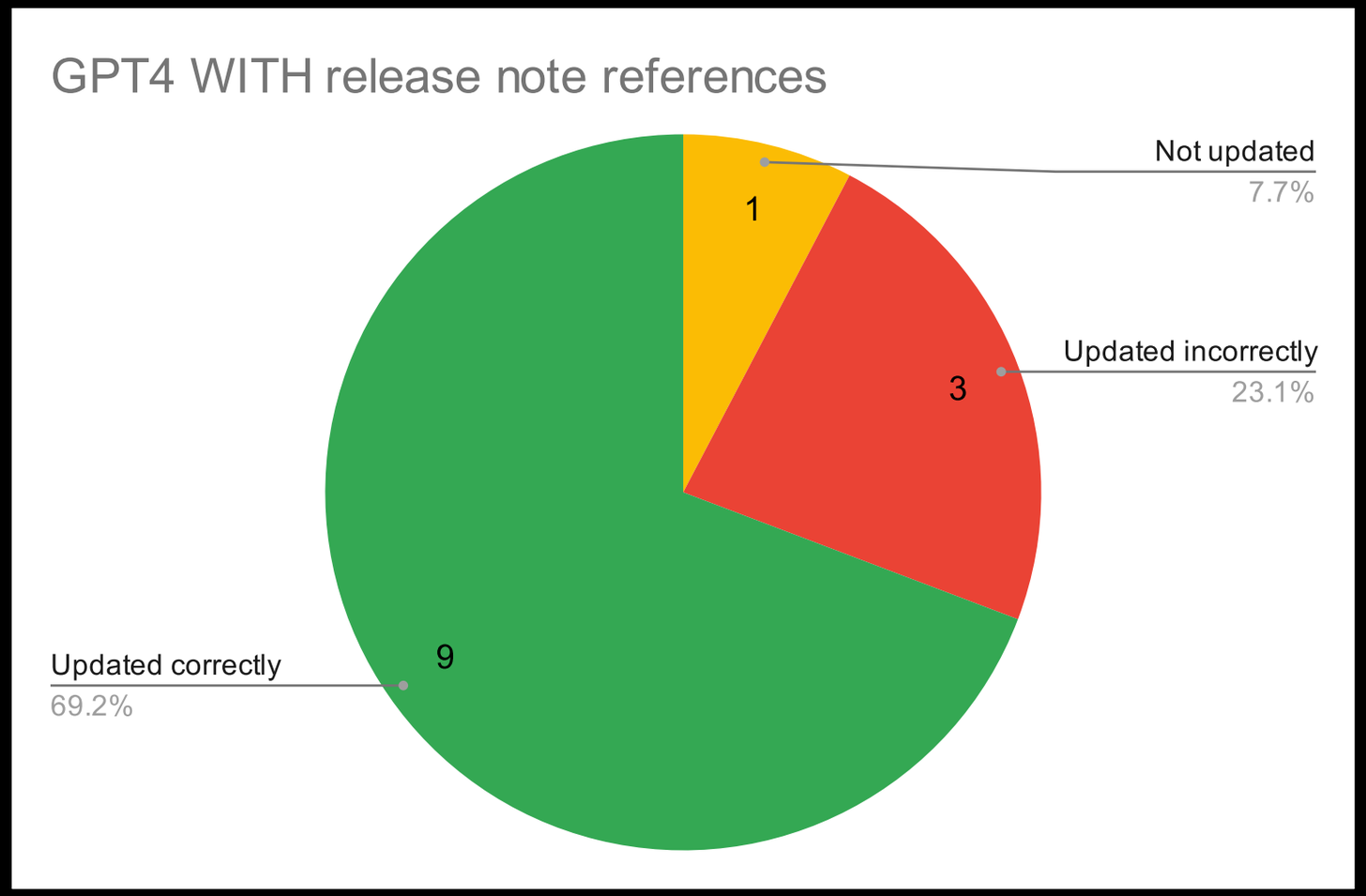


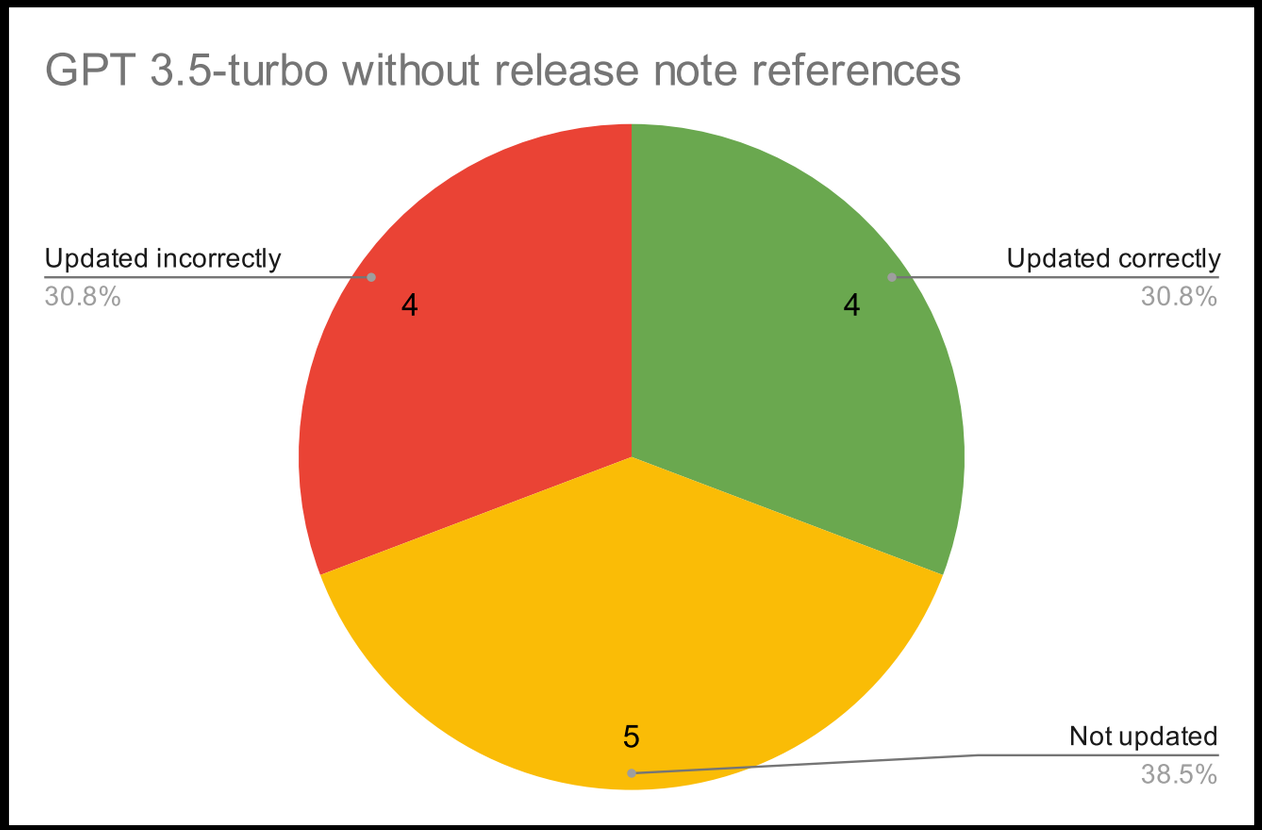
GPT 4

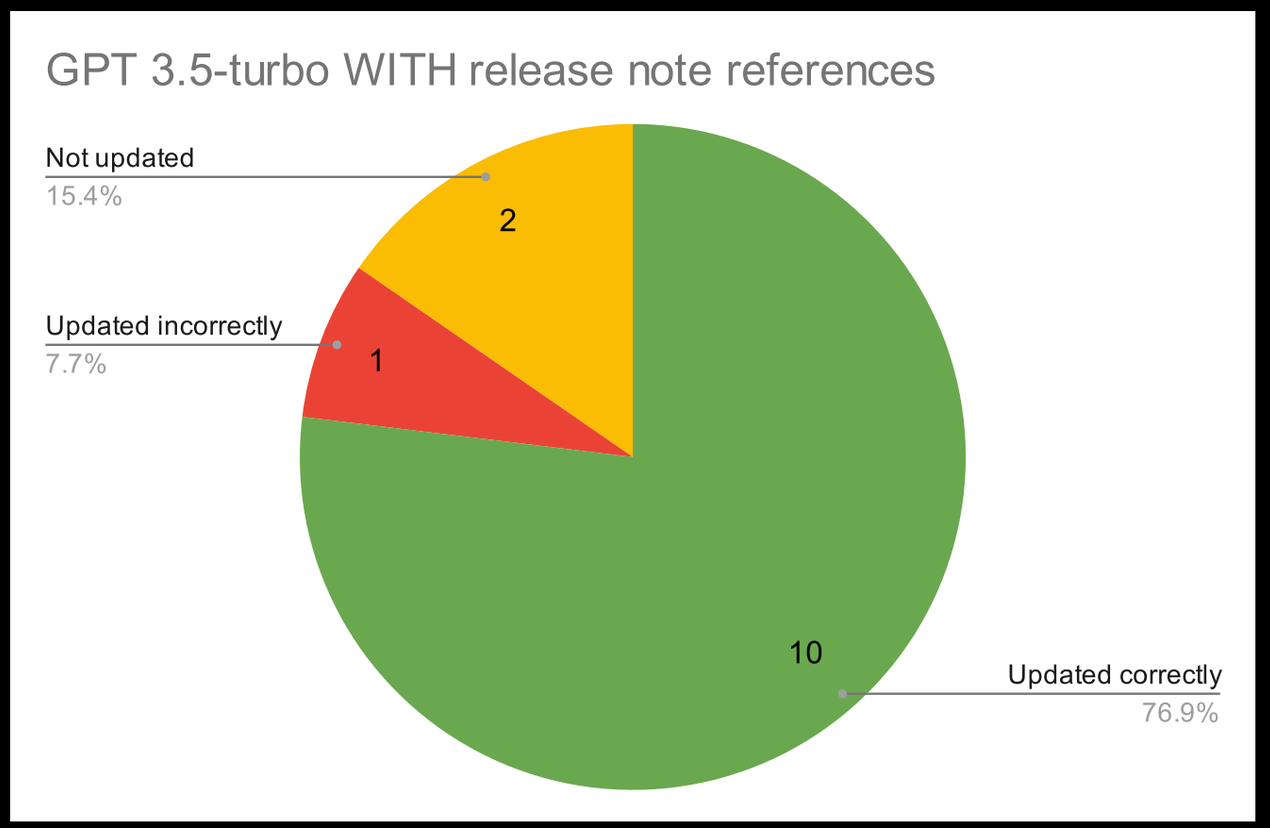
GPT4 alone always tries to update the code, resulting in lots of incorrect updates.

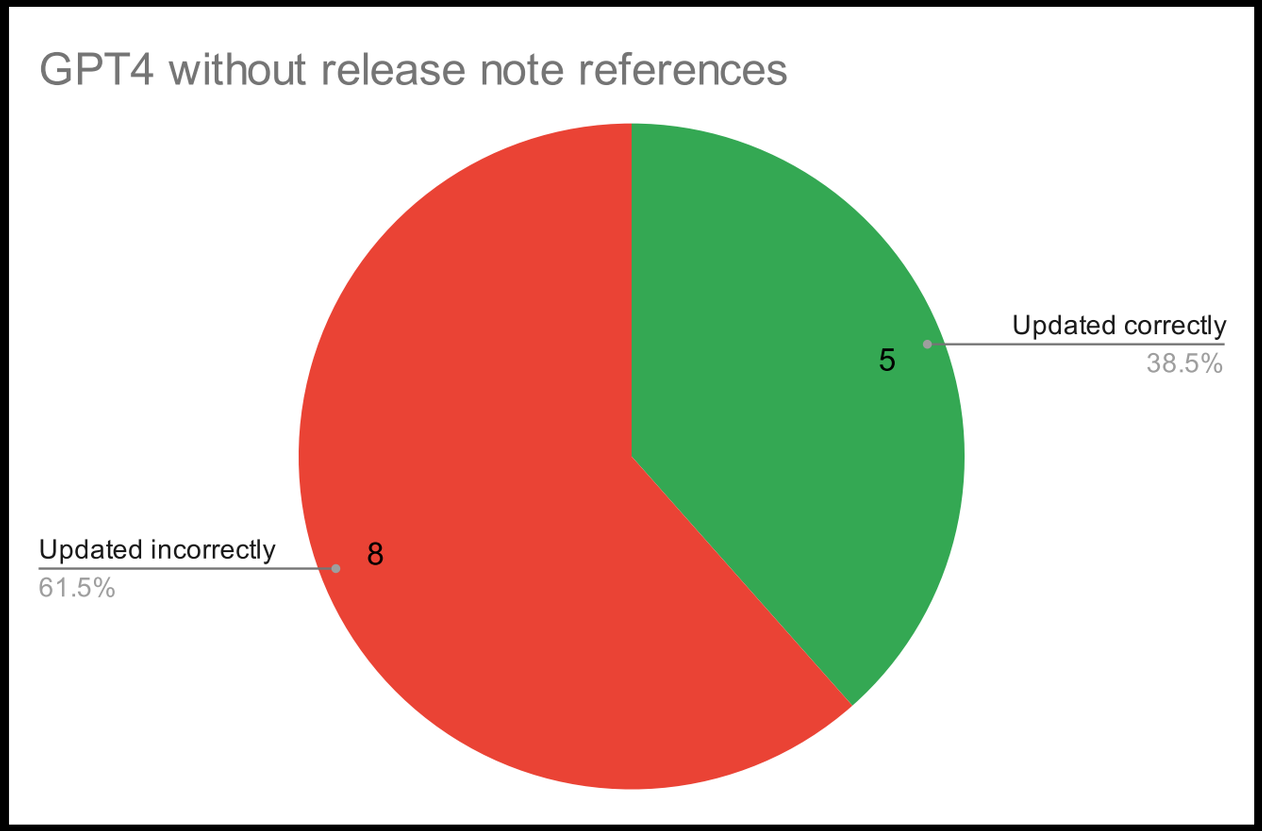


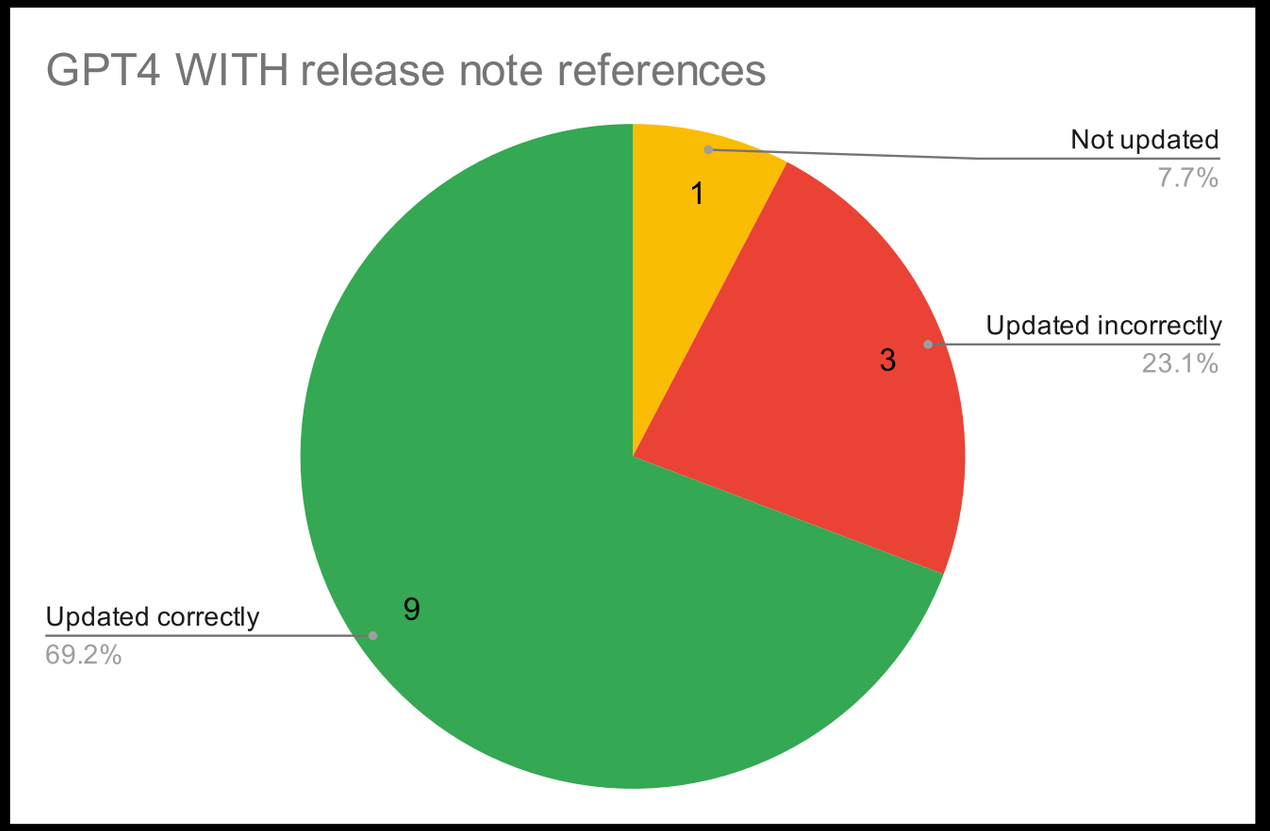
Adding references “grounds” it a bit, and results in more correct updates



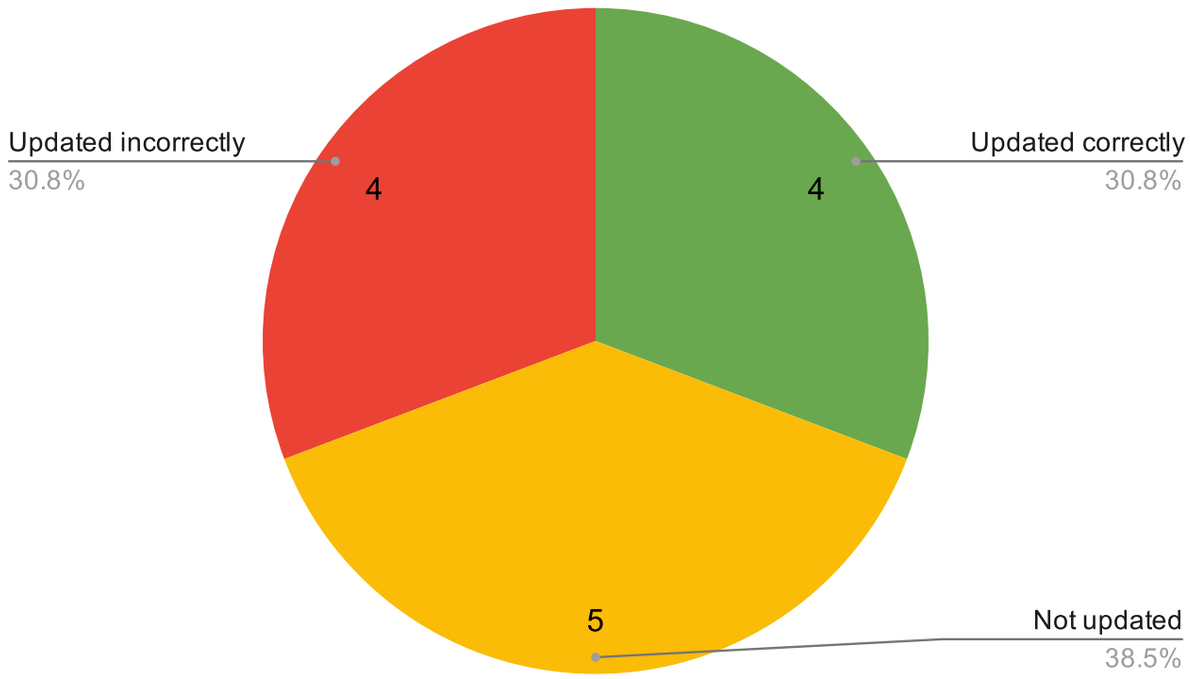

GPT 3.5-turbo without release note references
Updated incorrectly
30.8%
4
Updated correctly
30.8%
4
5
Not updated
38.5%


GPT 3.5-turbo WITH release note references
Not updated
15.4%
2
Updated incorrectly
7.7%
1
10
Updated correctly
76.9%

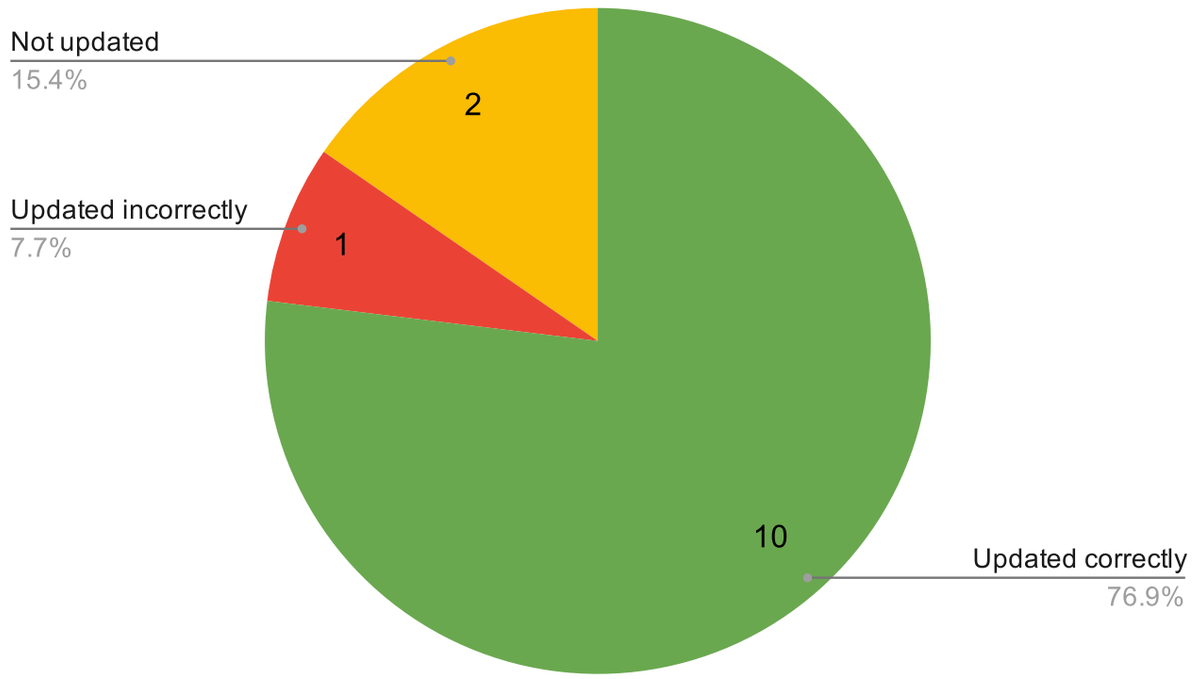

GPT4 without release note references
Updated correctly
38.5%
5
Updated incorrectly
61.5%
8


GPT4 WITH release note references
Not updated
7.7%
1
Updated incorrectly
23.1%
3
Updated correctly
69.2%
9

GPT 3.5-turbo without release note references

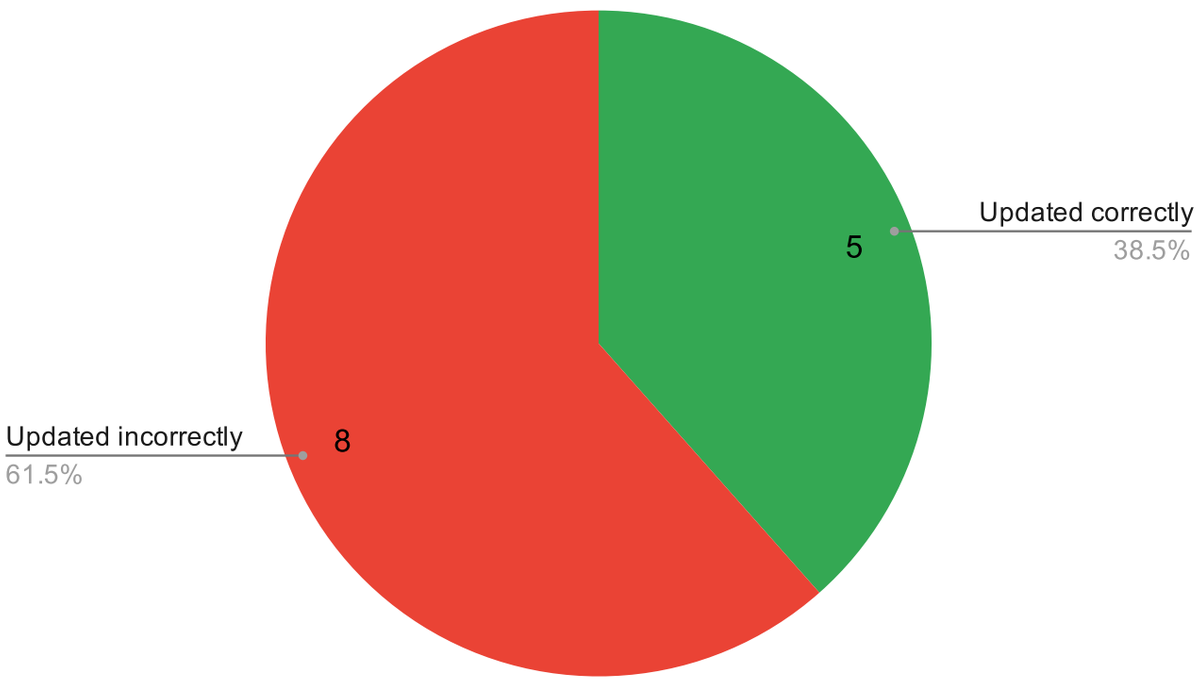


GPT 3.5-turbo WITH release note references

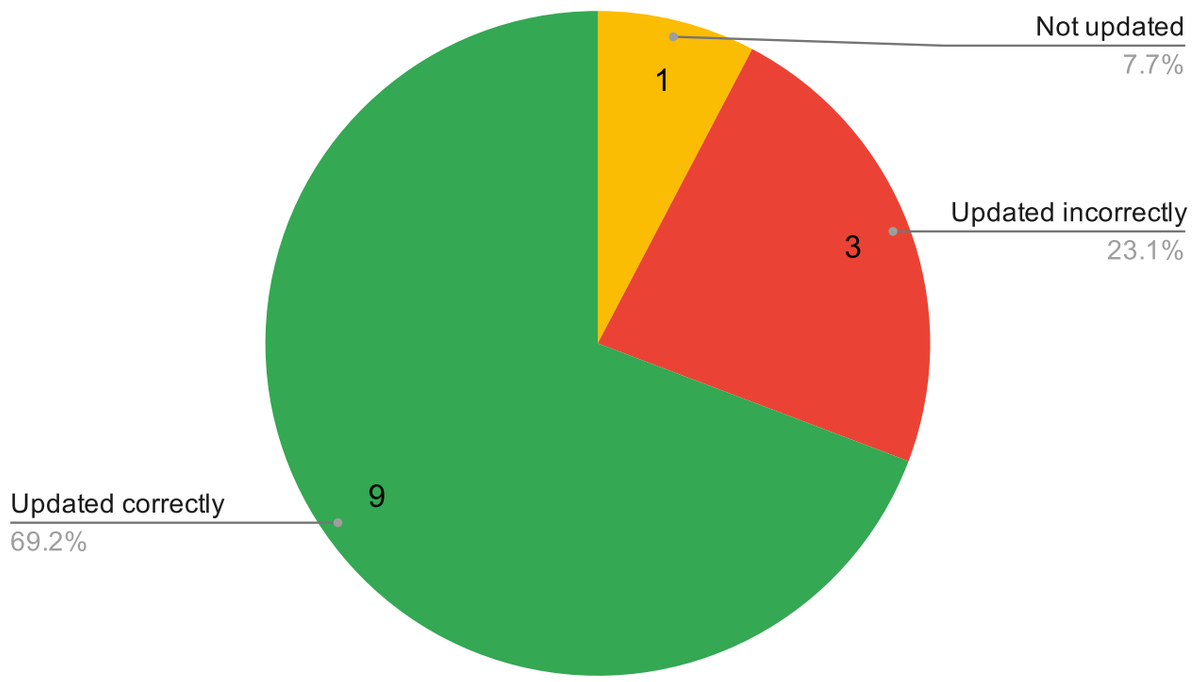


gpt4 does not necessarily result in improvement for every example

GPT4 without release note references



GPT4 WITH release note references



```python
import networkx as nx
import numpy as np

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5])
G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)])
matrix = nx.to_numpy_matrix(G)
print(matrix)
```

No references

…. [lots of irrelevant references] …

```python
import networkx as nx
import numpy as np

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5])
G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)])
matrix = nx.to_numpy_matrix(G)
print(matrix)
```

No references

…. [lots of irrelevant references] …

Both gpt3.5-turbo & gpt4 updated correctly

```python
import networkx as nx
import numpy as np

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5])
G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)])
matrix = nx.to_numpy_matrix(G)
print(matrix)
```

No references

Both gpt3.5-turbo & gpt4 updated correctly

…. [lots of irrelevant references] …

Gpt3.5-turbo still updated it correctly (providing the right reason but an irrelevant reference number) while gpt4 provided a somewhat incorrect update

```python
import networkx as nx
import numpy as np

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5])
G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)])
matrix = nx.to_numpy_matrix(G)
print(matrix)
```

No …] …

Both … rectly … t an … while … orrect … up …

Possible response:

```
import networkx as nx
import numpy as np

G = nx.Graph()
G.add_nodes_from([1, 2, 3, 4, 5])
G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)])
matrix = np.asarray(nx.to_numpy_matrix(G)) # use np.asarray instead of np.matrix
print(matrix)
```

Reason for update: The np.matrix class is deprecated and should be replaced by regular ndarrays. See https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

List of reference numbers used: No references used

Possible Next Steps

Possible Next Steps

- Evaluate this with more examples

Possible Next Steps

- Evaluate this with more examples
- Integrate in Copilot labs but w/ gutter notification on Copilot suggestions
- Provides data on how many Copilot suggestions need updating
- Provides explicit user feedback on accepting suggestions

Possible Next Steps

- Evaluate this with more examples
- Integrate in Copilot labs but w/ gutter notification on Copilot suggestions
  - Provides data on how many Copilot suggestions need updating
  - Provides explicit user feedback on accepting suggestions
- Evaluate on other languages

Challenges

Challenges

- Creating benchmark examples is tedious

Challenges

- Creating benchmark examples is tedious
- Automated evaluation is difficult, esp. because the model can change the example to have no warnings/errors but have diff semantics

Challenges

- Creating benchmark examples is tedious
- Automated evaluation is difficult, esp. because the model can change the example to have no warnings/errors but have diff semantics
  - Take some “incorrect” change w/ a grain of salt because it’s my interpretation

Challenges

- Creating benchmark examples is tedious
- Automated evaluation is difficult, esp. because the model can change the example to have no warnings/errors but have diff semantics
    - Take some “incorrect” change w/ a grain of salt because it’s my interpretation
- Technique requires continuous updating of release note DB

Challenges

- Creating benchmark examples is tedious
- Automated evaluation is difficult, esp. because the model can change the example to have no warnings/errors but have diff semantics
  - Take some “incorrect” change w/ a grain of salt because it’s my interpretation
- Technique requires continuous updating of release note DB
- Retrieval will likely become worse as more examples are added to DB — pre filtering by library can help

General Observations

General Observations

- Very hard to get the model to respect the response format you specify

General Observations

- Very hard to get the model to respect the response format you specify
- Even without giving it references, the model alone sometimes hallucinates reference numbers

General Observations

- Very hard to get the model to respect the response format you specify
- Even without giving it references, the model alone sometimes hallucinates reference numbers
- It's not clear how the model already knows about certain deprecations (but then updates them incorrectly anyways) — what is the model's "knowledge"

Automatically Updating Deprecated API Usage through LLMs and documentation retrieval

Sarah Nadi & Max Schaefer

Proposed Technique