Example Tasks

Example 1:

Repo URL	**https://github.com/albumentations-team/albumentations**
PR URL	**https://github.com/albumentations-team/albumentations/pull/2207**
Commit URL	**https://github.com/albumentations-team/albumentations/commit/cd4173b97b527ecf40097e55a18ffa7593b9f8fa**
Coding Language	**Python**

Write your Cursor question *

How does Albumentations handle spatial transforms (like Affine or ElasticTransform) on volumetric (3D) data, and what key parameters ensure consistency across slices in medical volumes? Provide a 3D augmentation pipeline example.

34 words

Write the full path file. *

You should write the full path file names of all the files mentioned in the prompt. If there are no file names just write “N/A”.

- Write this in a dictionary. If your files are, for example: @_typing.py and @merge.py, write:

{

“@_typing.py”: “pandas/_typing.py”,

“@merge.py”: “pandas/core/reshape/merge.py”

}

N/A

2 words

Claude 3.7 Sonnet Response

I'll look into how Albumentations handles spatial transforms for 3D data and provide an example pipeline for medical volumes.
Ran tool
Ran tool
Search files...
Based on my research, I can now provide a comprehensive explanation of how Albumentations handles spatial transforms for 3D volumetric data, with a focus on ElasticTransform and ensuring consistency across slices in medical volumes.

Albumentations 3D Spatial Transforms for Medical Volumes

How Albumentations Handles 3D Spatial Transforms

Albumentations applies spatial transforms to 3D volumes in one of two ways:

1. **2D transforms applied slice-wise**: Standard 2D spatial transforms like `ElasticTransform` and `Affine` are applied to each XY slice independently, but with identical parameters across all slices to maintain consistency through the Z-axis.

2. **Dedicated 3D transforms**: Specialized transforms like `CenterCrop3D`, `RandomCrop3D`, `CoarseDropout3D`, and `CubicSymmetry` that operate on the entire volume as a single 3D entity.

Key Parameters for Consistency in ElasticTransform

The `ElasticTransform` is particularly useful for medical imaging as it can simulate realistic anatomical variations. When applied to 3D volumes, these parameters ensure consistency:

```python
ElasticTransform(
    alpha=1.0,        # Controls deformation magnitude
    sigma=50.0,       # Controls smoothness of deformation (higher = smoother)
```

    interpolation=1, # Interpolation method (cv2.INTER_LINEAR=1)
    approximate=False,# Use precise method for displacement fields
    same_dxdy=False, # Whether to use same random field for both x and y
    noise_distribution="gaussian" # "gaussian" (natural) or "uniform" (mechanical)
)
```

### Critical Parameters for Medical Volumes:

1. **`sigma`**: Controls the smoothness of deformations. For medical volumes, higher values (50-100) create more anatomically plausible deformations.

2. **`alpha`**: Controls deformation magnitude. For medical data, conservative values (0.5-2.0) prevent unrealistic distortions.

3. **`same_dxdy`**: When set to `True`, uses the same displacement field for both X and Y directions, which can create more coherent deformations.

4. **`noise_distribution`**: "gaussian" produces more natural, anatomically plausible deformations compared to "uniform".

## 3D Augmentation Pipeline Example

Here's a complete example of a 3D augmentation pipeline for medical volumes:

```python
import albumentations as A
import numpy as np

def create_3d_medical_pipeline(
    crop_size=(64, 128, 128),
    p_spatial=0.5,
    p_intensity=0.3
):
    return A.Compose([
        # Memory efficiency: crop first
        A.RandomCrop3D(
            size=crop_size,
            p=1.0
        ),

        # Spatial transforms with consistency
        A.Flip(0, p=0.5),  # Flip along z-axis
        A.Flip(1, p=0.5),  # Flip along y-axis
        A.Flip(2, p=0.5),  # Flip along x-axis
        A.RandomRotate90((1, 2), p=0.5),  # Rotate in xy plane

        # ElasticTransform with parameters tuned for medical volumes
        A.ElasticTransform(
            alpha=1.5,  # Moderate deformation magnitude
            sigma=50.0,  # Smooth deformations
            approximate=False,  # Use precise method

```
            same_dxdy=True,  # More coherent deformations
            noise_distribution="gaussian",  # Natural-looking deformations
            p=0.3
        ),

        # 3D-specific transforms
        A.CubicSymmetry(p=0.2),  # Applies one of 48 possible 3D symmetry transforms

        # Intensity transforms (applied per slice with same parameters)
        A.RandomBrightnessContrast(p=0.3),
        A.RandomGamma(gamma_limit=(80, 120), p=0.3),

        # Simulate occlusions in 3D
        A.CoarseDropout3D(
            num_holes_range=(2, 5),
            hole_depth_range=(0.05, 0.15),  # Smaller holes along z-axis
            hole_height_range=(0.1, 0.2),
            hole_width_range=(0.1, 0.2),
            p=0.2
        ),
    ])

# Example usage
volume = np.random.rand(96, 256, 256).astype(np.float32)  # 3D volume (D,H,W)
mask = np.zeros((96, 256, 256), dtype=np.uint8)  # Segmentation mask

transform = create_3d_medical_pipeline()
transformed = transform(volume=volume, mask3d=mask)
transformed_volume = transformed['volume']
transformed_mask = transformed['mask3d']
```

Key Implementation Details

1. **Data Format**: Albumentations expects 3D volumes as numpy arrays with shape `(D, H, W)` or `(D, H, W, C)` where D is depth, H is height, W is width, and C is channels.

2. **Slice-wise Consistency**: When 2D transforms (like `ElasticTransform`) are included in a pipeline with 3D data, Albumentations samples their random parameters *once* per call and applies them identically to each XY slice along the depth axis.

3. **Memory Management**: Processing large 3D volumes can be memory-intensive. It's recommended to crop volumes to a manageable size at the beginning of the pipeline to reduce memory usage.

4. **Multiple Targets**: Transformations are synchronized between the volume and its corresponding mask3d, ensuring that spatial relationships are preserved.

5. **Performance Considerations**: ElasticTransform is computationally expensive for large volumes. Consider using `approximate=True` for faster processing when working with large volumes.

By carefully configuring these parameters, you can create realistic augmentations for medical imaging data while maintaining anatomical plausibility and consistency across slices.

Critical Rubrics - Response Successes. *

These are the critical rubrics that Claude 3.7 successfully achieved with a “YES”.

1. Does the response mention that `ElasticTransform` in 3D requires adjusting sigma and alpha to avoid non-physiological distortions in medical data?

24 words

Critical Rubrics - Response Fails *

These are rubrics that Claude 3.7 DID NOT have.

2. Does the response state that when applying `Affine` to a 3D volume, the same transformation matrix (including rotation and scale) is used for each individual slice to maintain anatomical consistency?

3. Does the response include a comparison of the performance of Affine3D vs. Affine (2D) in terms of images/second?

55 words

↪ Why were these rubrics correct?

Non-Critical Rubrics - Response Successes. *

These are rubrics that Claude 3.7 successfully achieved.

4. Does the response explain that it is required to handle `mask3D` in transforms that modify geometry (e.g., Rotate), ensuring that 3D annotations do not become misaligned?

5. Does the response clarify that 3D volumes must have shape (D, H, W, C) or (D, H, W) depending on whether they are multichannel or not?

59 words

Non-Critical Rubrics - Response Fails *

These are rubrics that Claude 3.7 answer DID NOT achieve.

6. Does the response provide an example of a 3D pipeline with Compose that combines `RandomRotate90`, `GaussianBlur3D`, and `RandomGamma` for CT data?

7. Does the response suggest using `cv2.INTER_LINEAR` for interpolating CT scans and `cv2.INTER_NEAREST` for MRI segmentation masks when applying `Resize3D`

8. Does the response mention interpolation method choices for medical data, specifically recommending `cv2.INTER_NEAREST` for masks and suggesting `cv2.INTER_CUBIC` over `cv2.INTER_LINEAR` for preserving anatomical detail?

9. Does the response mention whether using specific border modes such as `cv2.BORDER_REFLECT` changes the behavior of 3D spatial transforms like `ElasticTransform` when operating near the edges of the volume?

10. Does the response mention Z-axis resolution differences in medical volumes and suggest either scaling `ElasticTransform` parameters accordingly or preprocessing to make voxels isotropic?

152 words

↪ **Why were these rubrics correct?**

Supporting Context for the Inclusion of the Rubrics
1. The README does not detail specific parameters for 3D. In the code (transforms_3d.py, line 120), sigma controls the smoothness of the deformation, but there are no guidelines for values in medical domains. 2. The README specifies that spatial-level transforms maintain consistency in volumes (Spatial-level transforms section). The code in albumentations/core/transforms_3d.py shows that a single transformation matrix is used for all slices (lines 45–52). 3. The benchmark table in the README only includes 2D data. An expert might infer that 3D operations are ~3–5×

slower (because they process multiple slices), but this is not documented.

4. The README mentions that 3D masks are supported, but it does not explain strategies to preserve alignment during complex rotations (useful in tumor segmentation).

5. The README shows 2D Compose examples (line 12). In the official docs, it is suggested to use 3D transforms analogous to 2D ones, but no concrete examples are given.

6. The 3D transforms section of the README specifies input formats but does not highlight that channels (C) are optional. This is relevant to avoid preprocessing errors.

7. The source code (transforms_3d.py, line 88) allows configuration of interpolation, but the documentation provides no guidance on choosing optimal methods for medical imaging.

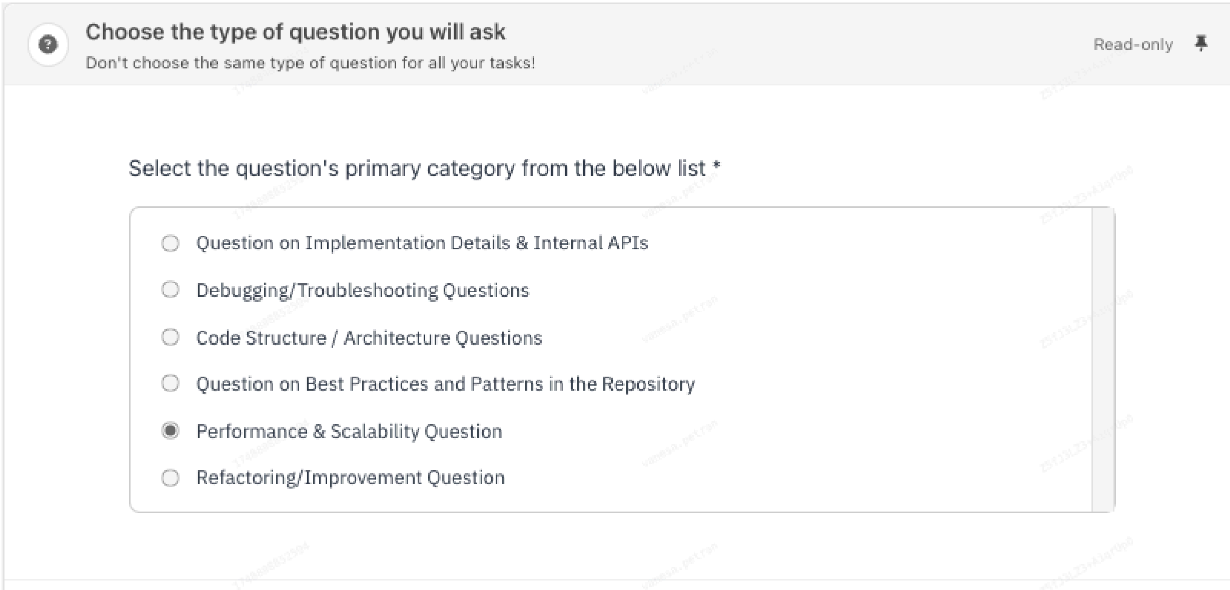
8. The code shows `mask_interpolation` parameter exists in `ElasticTransform` (transforms.py:201) where INTER_NEAREST prevents invalid mask values.

9. `ElasticTransform` uses border modes for displacement field application, affecting edge behavior in medical volumes (transforms.py:292-327).

10. Medical volumes often have anisotropic voxels (different Z vs XY resolution), requiring parameter adjustments for anatomically correct deformations.

Example 2:

The following is an example of a Good task submitted in the Ballerina Capuchina project ⭐



Choose the type of question you will ask
Don't choose the same type of question for all your tasks!

Read-only

Select the question's primary category from the below list *

- ○ Question on Implementation Details & Internal APIs
- ○ Debugging/Troubleshooting Questions
- ○ Code Structure / Architecture Questions
- ○ Question on Best Practices and Patterns in the Repository
- ● Performance & Scalability Question
- ○ Refactoring/Improvement Question

↪ **Why was this correct?**

While we need a diverse set of types of questions, many attempters focus on only providing Code Structure questions. Choosing a wide variety of questions across the tasks is important to maintain data variation in the dataset.

Write a question that will help you understand the repository. You can use the problem in the PR for inspiration of methods and files you may ask about. You may use Cursor to help you.

The questions should ALWAYS be related to the repository you are working with. You can ask the following types of questions: - Question on Implementation Details & Internal APIs - Debugging/Troubleshooting Questions - Code Navigation/ Structure/ Design Questions - Questions on Best Practices and Patterns - Performance & Scalability Questions - Refactoring/Improvement Questions

Read-only

Write your Cursor question *

What is the impact of replacing the in-memory update logic with a persistent traversal-based index updater in the keyValueIndexed database on the consistency, performance and modularity of the OrbitDB system?

35 words

Write the full path file. *

You should write the full path file names of all the files mentioned in the prompt. If there are no file names just write "N/A".

Example:

- @FloatingPanel.tsx --> packages/vant/src/floating-panel/FloatingPanel.tsx

src/databases/keyvalue-indexed.js

5 words

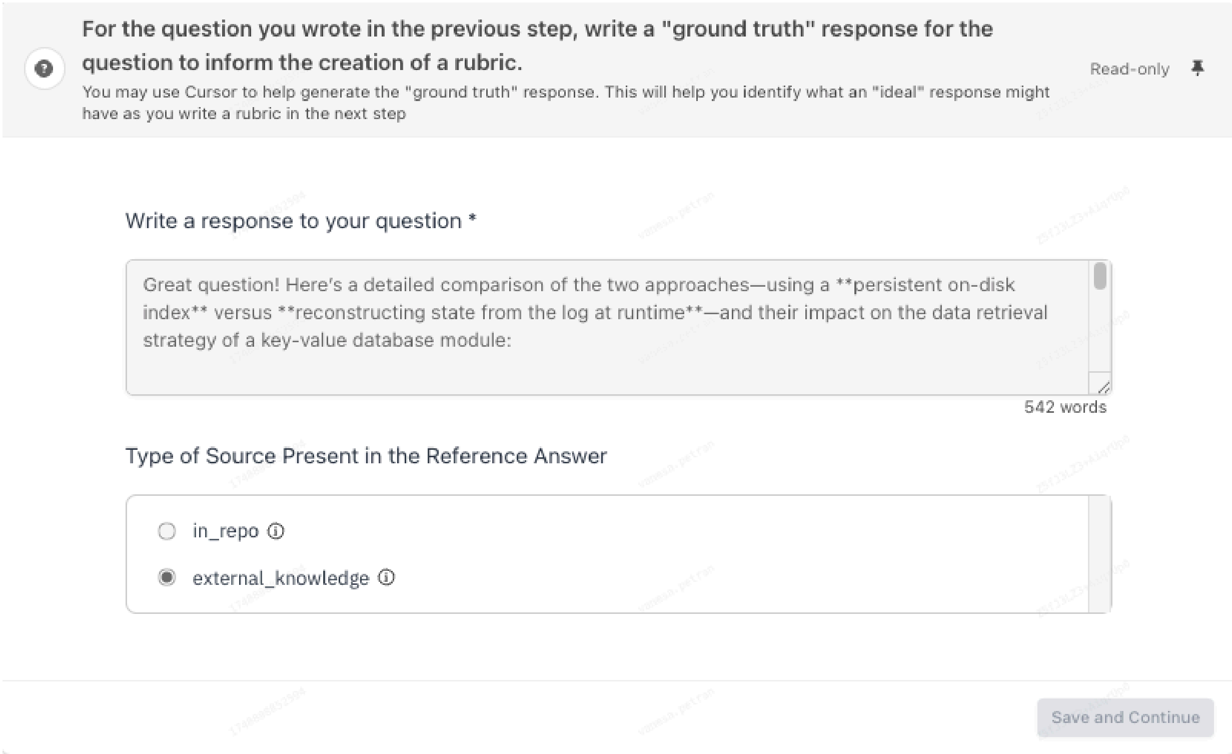
Save and Continue

↪ Why was this question correct?

The question is specific to the repository in the task. The question is complex enough so that even the strongest models might have an issue providing a complete answer to it.

↪ Are there any issues with this part of the task?

Yes, the contributor provided a path file when no file was directly mentioned in the prompt.



For the question you wrote in the previous step, write a "ground truth" response for the question to inform the creation of a rubric.

You may use Cursor to help generate the "ground truth" response. This will help you identify what an "ideal" response might have as you write a rubric in the next step

Read-only

Write a response to your question *

Great question! Here's a detailed comparison of the two approaches—using a **persistent on-disk index** versus **reconstructing state from the log at runtime**—and their impact on the data retrieval strategy of a key-value database module:

542 words

Type of Source Present in the Reference Answer

- in_repo
- external_knowledge

Save and Continue

↪ Why was this answer correct?

The provided answer was created using Claude 3.7 in Cursor. Providing GPT answers, or Claude answers not being crafted using Cursor will result in a bad quality task and a SBQ.

Write rubrics to evaluate potential model responses. The rubrics should target CRITICAL and NICE TO HAVE (non-critical) items.

These rubrics should be a checklist of items that an "ideal" model response meets. The reference answer should FAIL +40% of the rubrics you write.

Read-only

The answer for this field is already provided.

Critical Rubrics - Response Successes. *

These are the critical rubrics that the reference answer successfully achieved with a "YES".

1. Does the response explain that using a persistent traversal-based index in `src/databases/keyvalue-indexed.js` introduces a trade-off between slower write performance due to disk I/O and faster crash recovery by avoiding full index rebuilds?

2. Does the response describe that modularity improves in `KeyValueIndexed` when indexing logic is decoupled and implemented as a separate updater module, enabling strategies like plugin-based design?

70 words

Critical Rubrics - Response Fails *

These are rubrics that the reference answer DID NOT have.

3. Does the response explain that the persistent index updater integrates with the `onUpdate` lifecycle callback used by `KeyValue()` in `src/databases/keyvalue-indexed.js`?

4. Does the response specify that consistency is maintained or ensured when `Index.update` processes log entries in partially failed states?

5. Does the response address the issue of key-value retrieval performance from the persisted index remains stable in long-running systems, particularly after partial reindexing or crash recovery?

6. Does the response identify that memory usage is affected when transitioning from in-memory indexing to persistent log-based indexing in `KeyValueIndexed`?

↪ Why were these rubrics correct?

- **These rubrics are self-contained, atomic, and they do not overlap.**
- **The rubrics are specific to the code, and they do not only assess general programming concepts.**
- **Notice how the attempter provided a good number of Critical Rubrics fails.**
- **The formatting follows the instructions as the rubrics continue the numeration across the different fields.**

Non-Critical Rubrics - Response Successes. *

These are rubrics that the reference answer successfully achieved.

7. Does the response provide a comparative summary table outlining consistency, performance, and modularity trade-offs between the in-memory and persistent index approaches?

8. Does the response suggest a hybrid indexing approach that combines fast in-memory updates with periodic persistence to disk as a compromise solution?

48 words

Non-Critical Rubrics - Response Fails *

These are rubrics that the reference answer DID NOT achieve.

9. Does the code in `src/databases/keyvalue-indexed.js` show that the persistent traversal-based updater avoids reprocessing log entries by checking whether they have already been indexed?

10. Does the response mention whether the update function in the persistent index operates incrementally across sessions to avoid full index rebuilds after a restart?

11. Does the response acknowledge whether the persistent traversal-based updater introduces any additional dependencies or storage abstractions not present in the previous in-memory logic?

80 words

↪ Why were these rubrics correct?

- **These rubrics are self-contained, atomic, and they do not overlap.**
- **Notice how the attempter provided a good number of Non-Critical Rubrics fails.**
- **The rubrics are specific to the code, and they do not only assess general programming concepts.**
- **The formatting follows the instructions as the rubrics continue the numeration across the different fields.**

Supporting context for the inclusion of each rubric. You should map each rubric [1., 2., 3., ...] with the supporting information [1., 2., 3., ...] that justifies the inclusion of the rubric. *

Map the rubrics (1, 2, 3, ...) with an explanation with the part of the code (1, 2, 3, ...) that justifies the inclusion of that rubric... For each success code-related rubric, provide the file/line number/function declaration that justifies the rubric inclusion. For more conceptual rubrics, you can write a short justification for the rubric's inclusion in your rubric list.

1. They `keyvalue-indexed.js` file replaces the in-memory index with a persistent one (`Index.update`), which writes to disk (`await index.put`, `await indexedEntries.put`). This involves additional I/O during PUT/DEL operations. In return, the index no longer needs to be rebuilt from scratch after a crash.

2. The new `Index` is defined as a separate function and is explicitly passed to `onUpdate`, making it possible to replace this behaviour without touching the core of `KeyValueIndexed`.

3. In the line where `keyValueStore` is constructed, `onUpdate: index, update` is passed, showing how the updater is connected to the database operation flow. This establishes the direct relationship between writing entries and updating the index.

4. `Index.update` checks whether each hash has already been indexed, allowing it to continue safely and avoid duplication or inconistencies if the process is interrupted. It also skips entries if they have already been recorded, maintaining consistency without re-executing everything.

5. The storage of keys and values is achieved through the use of `index.input()` and `index.get()`. These accesses are consistent in time, which enables the maintenance of stability in systems that have endured multiple crashes. This is due to the fact that no state reconstruction from the entire log is necessary.

6. The index now persists data to disk (LevelStorage) instead of keeping it in volatile structures such as Map or Set. This reduces memory usage in long runs, transferring the load to the file system.

7. This heading is conceptual it is not inferred from the code, but is justified by the existence of two approaches that are alternatives for the same file (`_updateIndex` vs `Index.update`), which opens up space for a direct structural comparison.

8. The current code does not implement a hybrid approach, but the modular design, by passing `onUpdate` as a parameter would allow it.

9. Within `Index.update`, `indexedEntries.get(hash)` and `indexedEntries.put(hash, true)` are used to record which entries have already been processed, thus avoiding redundant rewrites.

10. By recording the processed hashes in `indexedEntries` the function can restart from the last entry without traversing the entire log again, allowing for incremental updating between sessions.

11. Unlike `_updateIndex` which used only one Map, the new updater uses two instances of `LevelStorage(index, indexedEntries)`, introducing a new layer of persistent storage.

↪ Why was the supporting context correct?

- **Each of the items maps directly to one of the numbered rubrics**
- **The attempter provides justifications as to why the inclusion of the rubrics is relevant.**