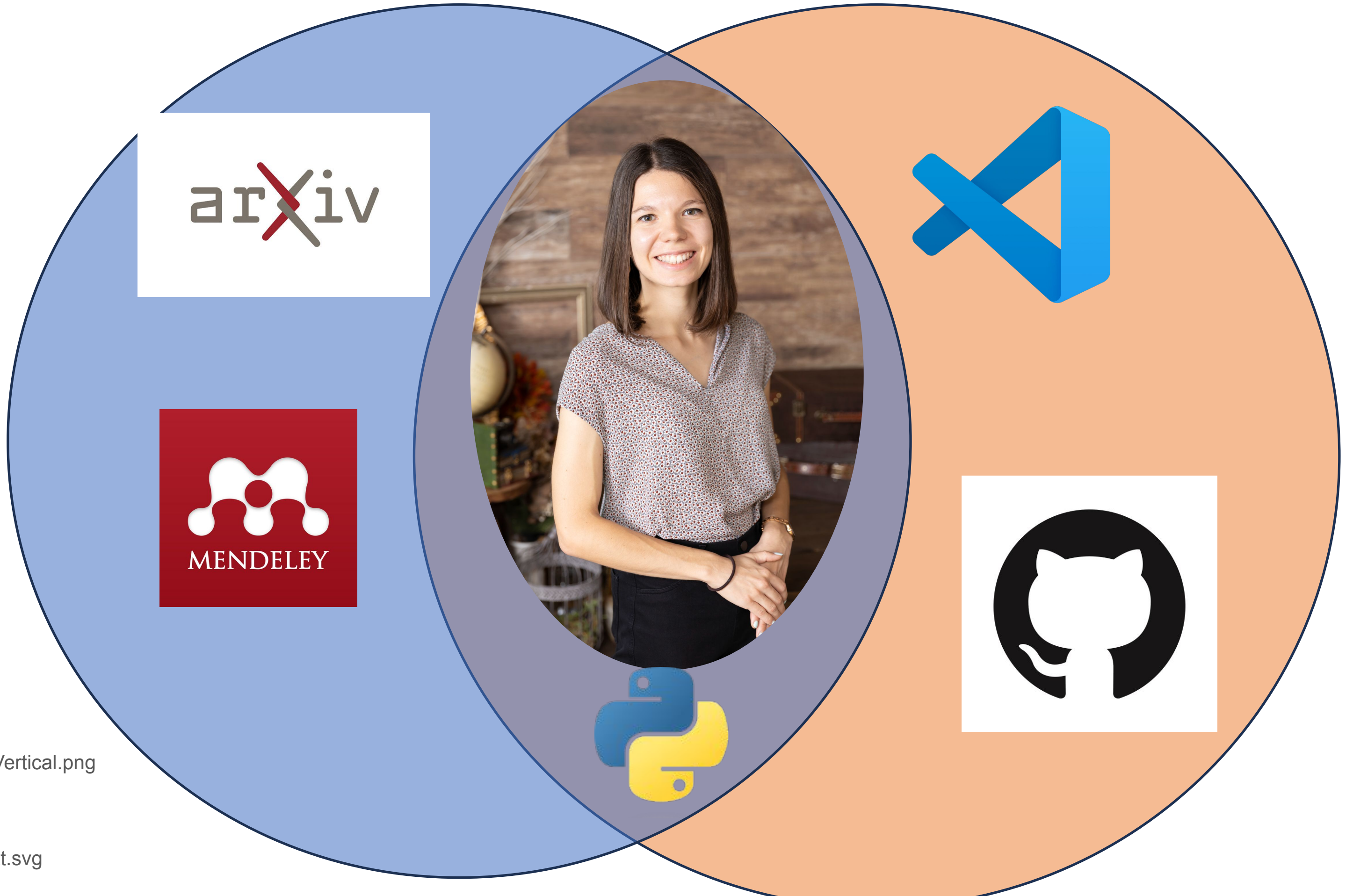
The Path to Good Software Hygiene

By Crafting Good Habits

Betty Le Dem

whoami

- Betty Le Dem, Machine Learning Engineer at Woven by Toyota.



https://en.m.wikipedia.org/wiki/File:Mendeley_Logo_Vertical.png
https://info.arxiv.org/brand/logos.html
https://github.com/logos
https://code.visualstudio.com/brand
https://en.m.wikipedia.org/wiki/File:Python-logo-notext.svg

Context Window

- Becoming good at coding is hard.
- Becoming better at coding is feasible.

I would like to share four sets of habits that helped me to be more efficient and confident in my code.

1. Stone Age Habits
2. Modern Habits
3. Cutting Edge Habits
4. Beyond Habits…

Stone Age Habits

Your codebase is your home.

- Be kind with yourself by cleaning your own space.
- Clean and tidy your room before inviting anyone.

Stone Age Habits

- Know your tools
  - Typing
  - Shortcuts
  - VScode
  - Oh my zsh
  - Docker
  - Git
  - Etc.

Modern Age Habits

- Keep learning (it’s easy to get outdated)
- Find your favorite learning resources
  - Books
  - Classes.
  - Open sources projects
  - Newsletters (Substack, Pragmatic Engineer, Import AI, The Batch)
  - Social Media (X, LinkedIn)

Modern Age Habits

- Learning the best practices from experienced practitioners.

Documentation

Flexibility

Continuous Integration

Automated Testing

Maintainability

Tests

Easy To Change

Scalability

Open-Close Principle

Don't Repeat Yourself

Version Control

Readability

Scalability

YAGNI

Modularity

Variable Naming

Refactoring

Encapsulation

SOLID

Code Standard

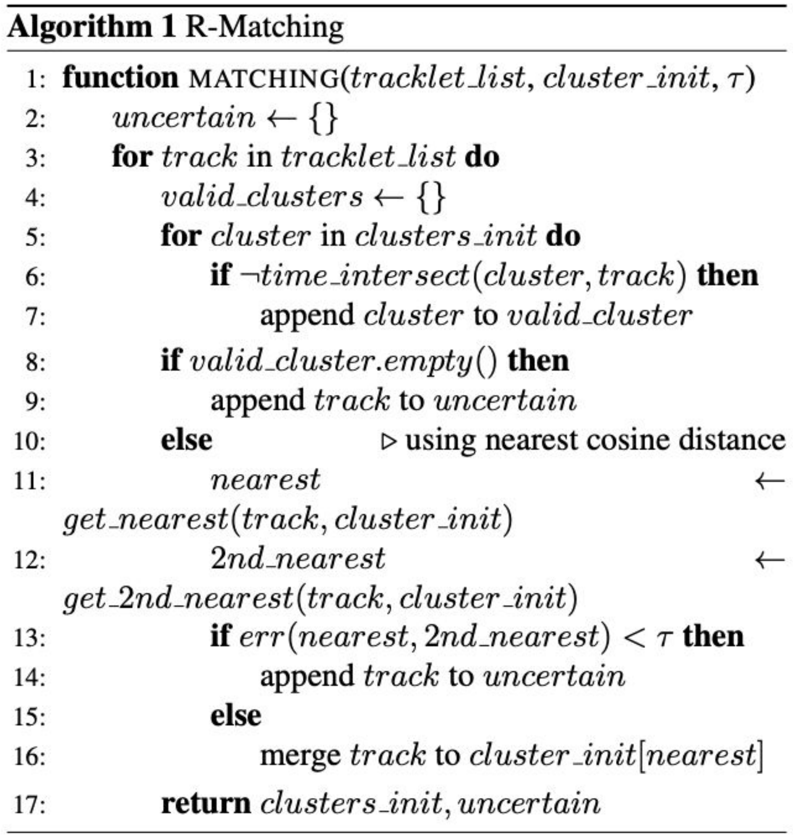
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
The Mythical Man-Month
Frederick P. Brooks, Jr.
Refactoring
Improving the Design of Existing Code
Martin Fowler
O'REILLY
Software Engineering at Google
Lessons Learned from Programming Over Time
The Pragmatic Programmer
David Thomas
Andrew Hunt
A PHILOSOPHY OF SOFTWARE DESIGN
John Ousterhout

Cutting Edge Habits

- Use AI to design, improve and clean your room.
  - Copilot
  - ChatGPT

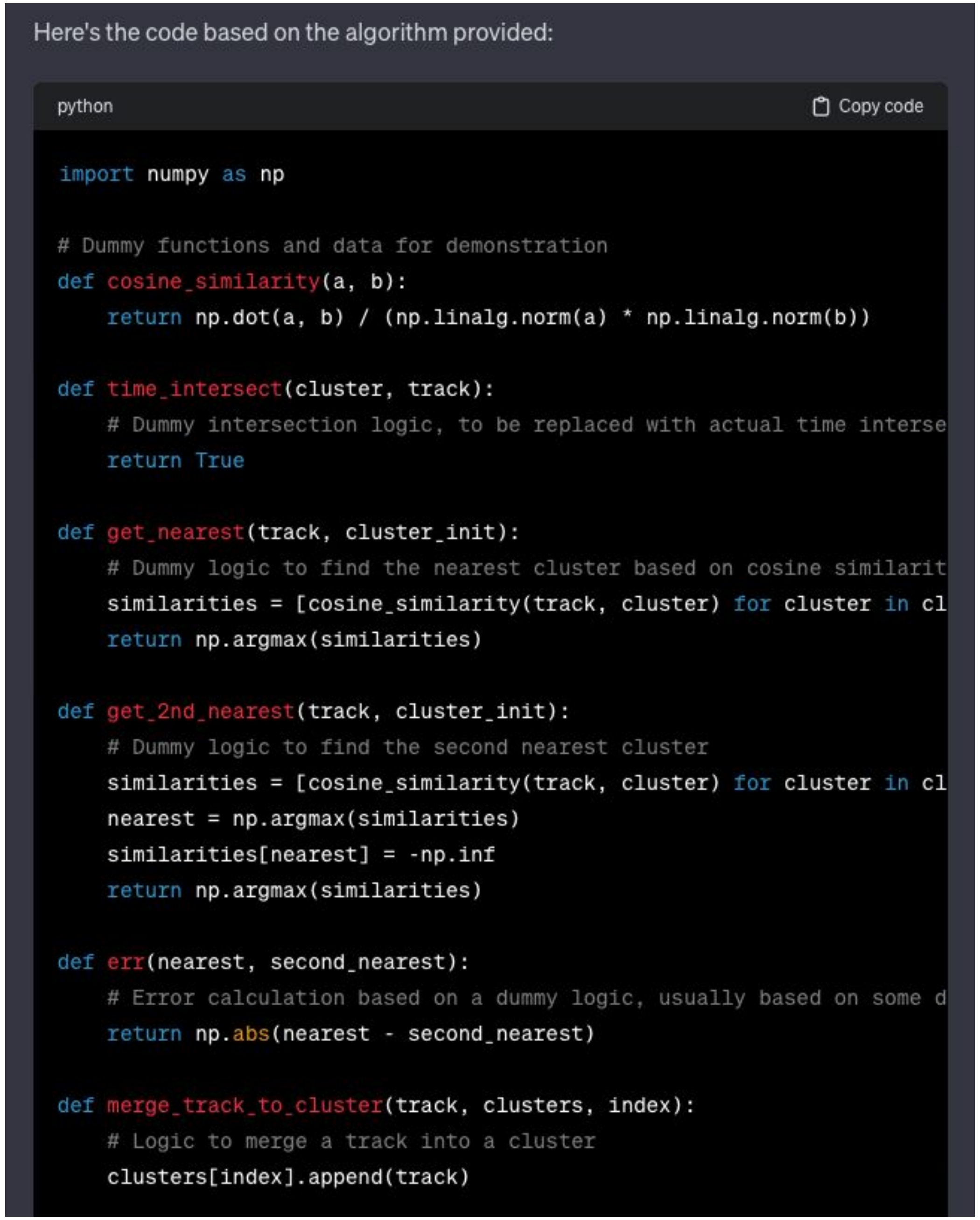
Demo GPT for code generation

“In real-life scenes, however, the feature model’s ability to discriminate between individuals with different features for inter-class persons is unclear, as it is trained on the synthetic person dataset. Figure 3b depicts the similarity score between two individuals wearing white shirts is 0.85. Hence, during clustering, inter-class persons may be incorrectly grouped together. To address this issue, we propose an algorithm that can overcome this issue. First, we select the frames with the highest number of people. Then we construct a graph for the individuals in each frame, with vertices representing people and edges determined by their cosine similarity. Then, we select the frame with the smallest sum of edges as the initial cluster. Finally, we employ the algorithm depicted in algorithm 1 to merge tracklets that fit into the same cluster.”

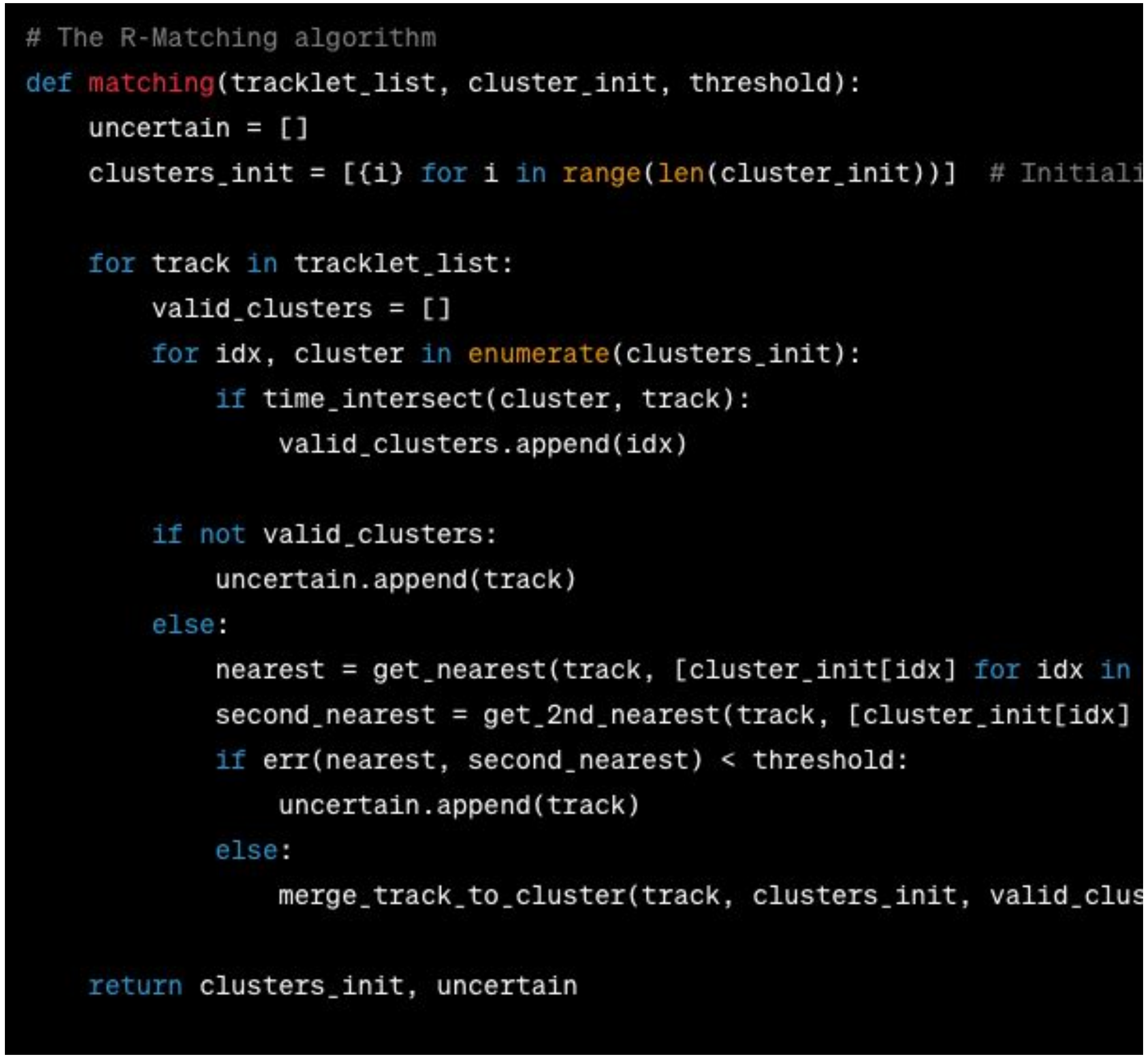


Algorithm 1 R-Matching

```
function MATCHING($tracklet\_list, cluster\_init, \tau$)
    $uncertain \leftarrow \{\}$
    for $track$ in $tracklet\_list$ do
        $valid\_clusters \leftarrow \{\}$
        for $cluster$ in $clusters\_init$ do
            if $\neg time\_intersect(cluster, track)$ then
                append $cluster$ to $valid\_cluster$
        if $valid\_cluster.empty()$ then
            append $track$ to $uncertain$
        else                                ▷ using nearest cosine distance
            $nearest \leftarrow get\_nearest(track, cluster\_init)$
            $2nd\_nearest \leftarrow get\_2nd\_nearest(track, cluster\_init)$
            if $err(nearest, 2nd\_nearest) < \tau$ then
                append $track$ to $uncertain$
            else
                merge $track$ to $cluster\_init[nearest]$
        return $clusters\_init, uncertain$
```


Here's the code based on the algorithm provided:

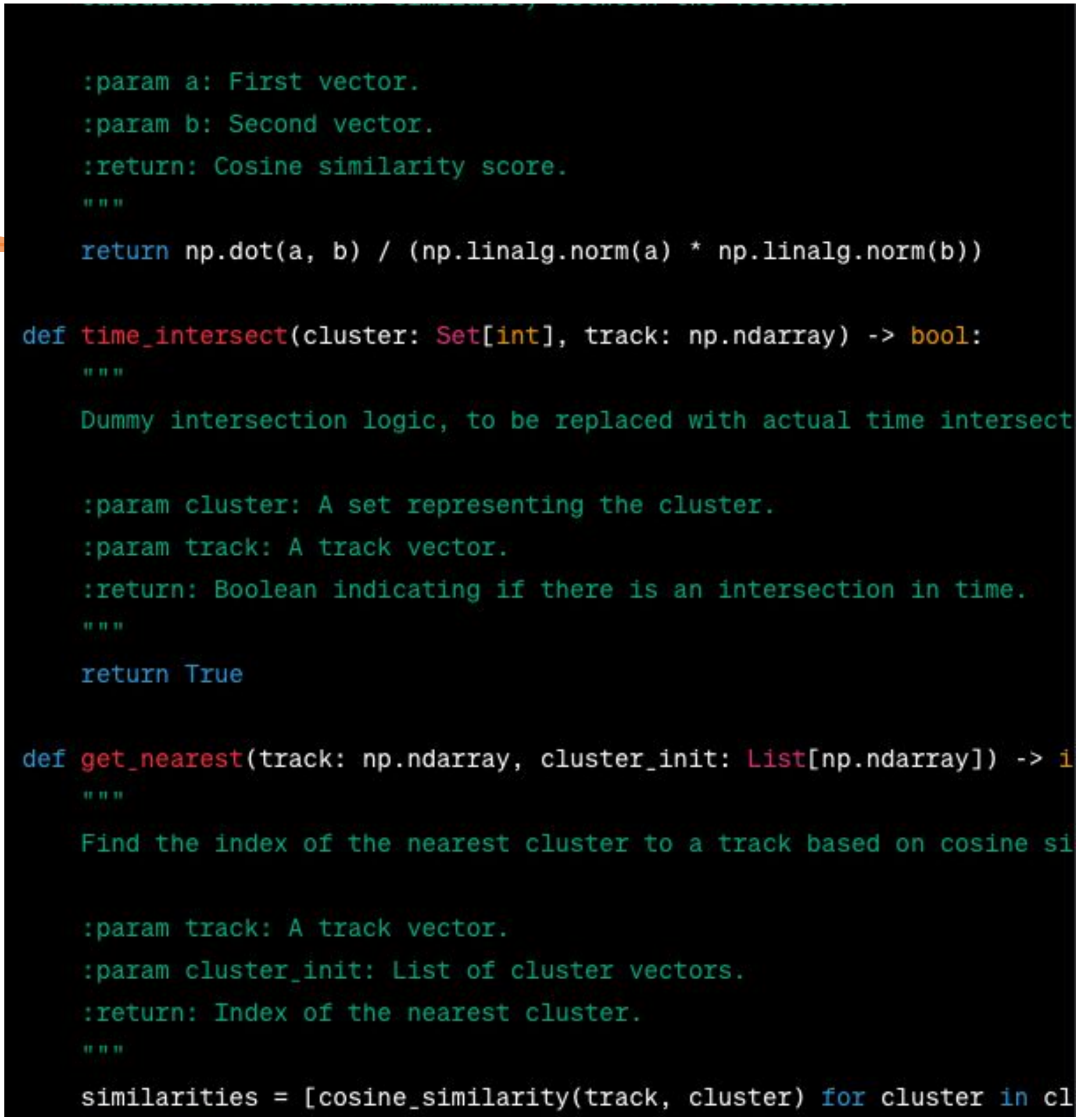
```python
import numpy as np

# Dummy functions and data for demonstration
def cosine_similarity(a, b):
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

def time_intersect(cluster, track):
    # Dummy intersection logic, to be replaced with actual time interse
    return True

def get_nearest(track, cluster_init):
    # Dummy logic to find the nearest cluster based on cosine similarit
    similarities = [cosine_similarity(track, cluster) for cluster in cl
    return np.argmax(similarities)

def get_2nd_nearest(track, cluster_init):
    # Dummy logic to find the second nearest cluster
    similarities = [cosine_similarity(track, cluster) for cluster in cl
    nearest = np.argmax(similarities)
    similarities[nearest] = -np.inf
    return np.argmax(similarities)

def err(nearest, second_nearest):
    # Error calculation based on a dummy logic, usually based on some d
    return np.abs(nearest - second_nearest)

def merge_track_to_cluster(track, clusters, index):
    # Logic to merge a track into a cluster
    clusters[index].append(track)
```



```python
# The R-Matching algorithm
def matching(tracklet_list, cluster_init, threshold):
    uncertain = []
    clusters_init = [{i} for i in range(len(cluster_init))]  # Initiali

    for track in tracklet_list:
        valid_clusters = []
        for idx, cluster in enumerate(clusters_init):
            if time_intersect(cluster, track):
                valid_clusters.append(idx)

        if not valid_clusters:
            uncertain.append(track)
        else:
            nearest = get_nearest(track, [cluster_init[idx] for idx in
            second_nearest = get_2nd_nearest(track, [cluster_init[idx]
            if err(nearest, second_nearest) < threshold:
                uncertain.append(track)
            else:
                merge_track_to_cluster(track, clusters_init, valid_clus

    return clusters_init, uncertain
```

Feedback loop

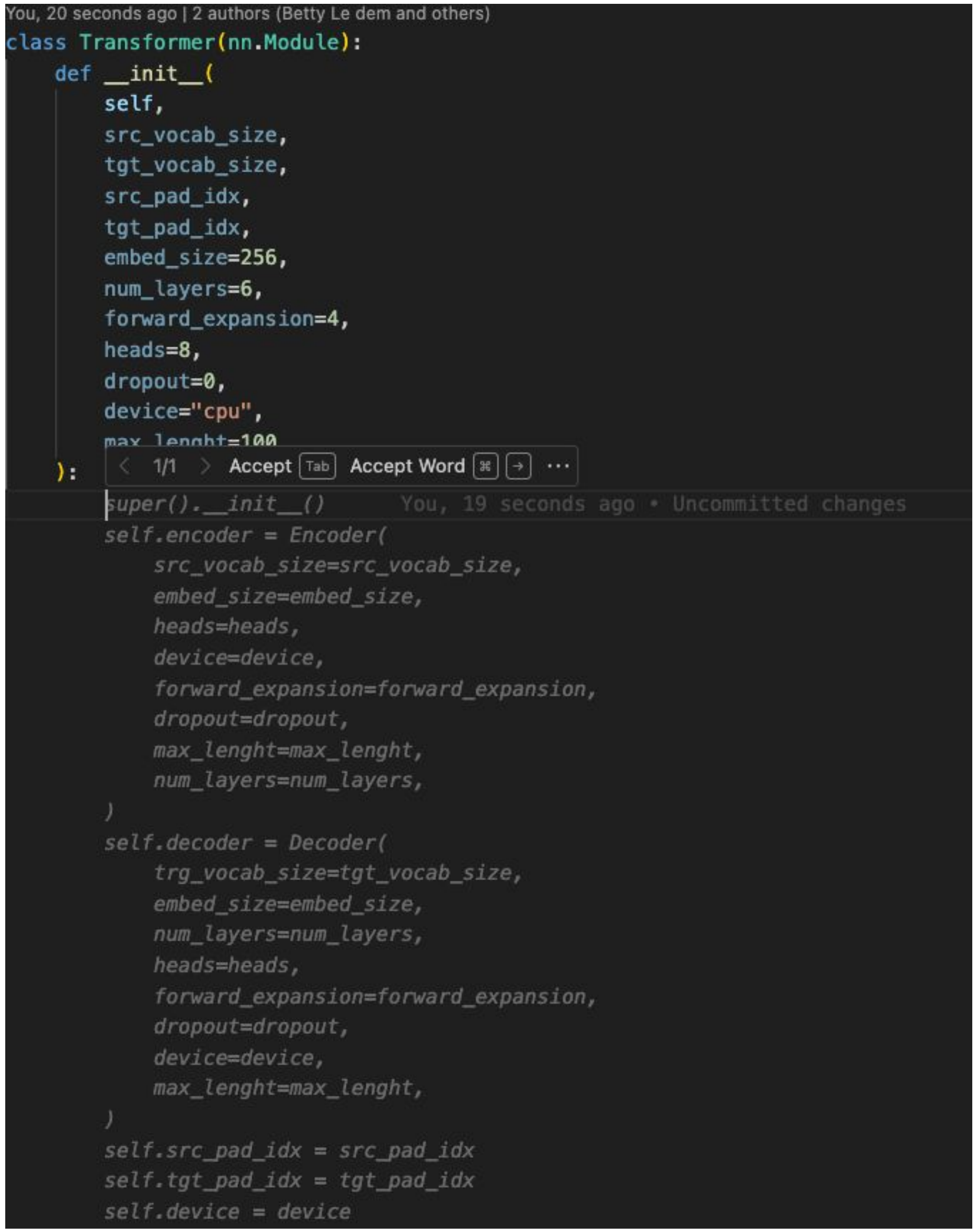
- “Add type hint and documentation”



```
    :param a: First vector.
    :param b: Second vector.
    :return: Cosine similarity score.
    """
    return np.dot(a, b) / (np.linalg.norm(a) * np.linalg.norm(b))

def time_intersect(cluster: Set[int], track: np.ndarray) -> bool:
    """
    Dummy intersection logic, to be replaced with actual time intersect

    :param cluster: A set representing the cluster.
    :param track: A track vector.
    :return: Boolean indicating if there is an intersection in time.
    """
    return True

def get_nearest(track: np.ndarray, cluster_init: List[np.ndarray]) -> i
    """
    Find the index of the nearest cluster to a track based on cosine si

    :param track: A track vector.
    :param cluster_init: List of cluster vectors.
    :return: Index of the nearest cluster.
    """
    similarities = [cosine_similarity(track, cluster) for cluster in cl
```

Demo Copilot

- Good at logic completion
- Provide insights of common practices in real time.
- Good at boilerplate code and completing from existing code.
- Good at picking up code style and propose customized suggestions.



```
class Transformer(nn.Module):
    def __init__(
        self,
        src_vocab_size,
        tgt_vocab_size,
        src_pad_idx,
        tgt_pad_idx,
        embed_size=256,
        num_layers=6,
        forward_expansion=4,
        heads=8,
        dropout=0,
        device="cpu",
        max_lenght=100
    ):
        super().__init__()
        self.encoder = Encoder(
            src_vocab_size=src_vocab_size,
            embed_size=embed_size,
            heads=heads,
            device=device,
            forward_expansion=forward_expansion,
            dropout=dropout,
            max_lenght=max_lenght,
            num_layers=num_layers,
        )
        self.decoder = Decoder(
            trg_vocab_size=tgt_vocab_size,
            embed_size=embed_size,
            num_layers=num_layers,
            heads=heads,
            forward_expansion=forward_expansion,
            dropout=dropout,
            device=device,
            max_lenght=max_lenght,
        )
        self.src_pad_idx = src_pad_idx
        self.tgt_pad_idx = tgt_pad_idx
        self.device = device
```

Beyond the habits

- A style which fits you.
- There is no wrong way to do something.
- Coding is fun.

Thank you!