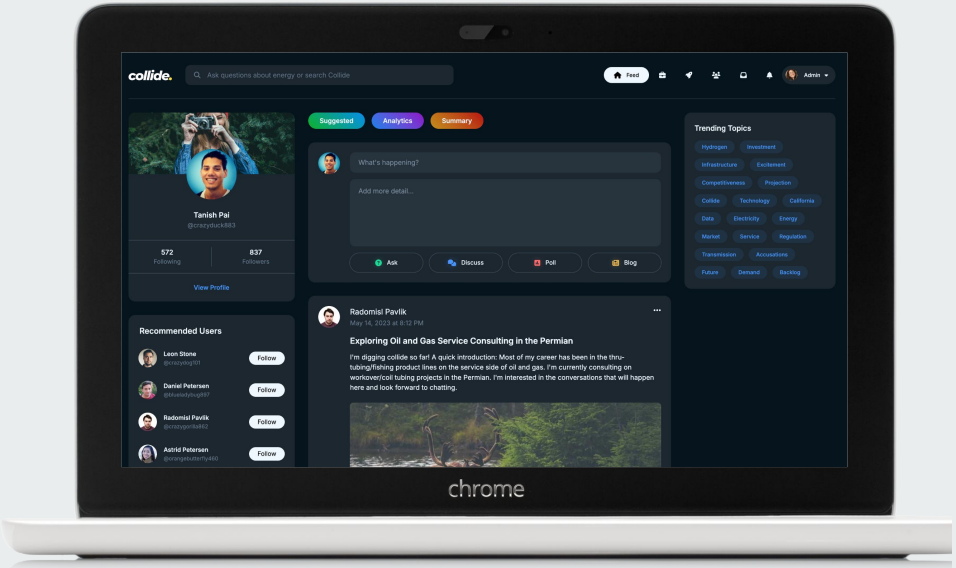
Collide Content Recommendation Engine

Team WorksOnMyMachine



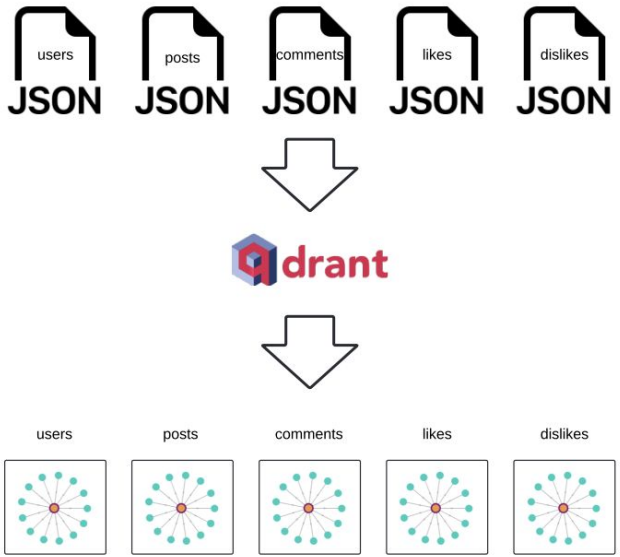
Stack

Qdrant database, Flask Backend, React Frontend, Open AI

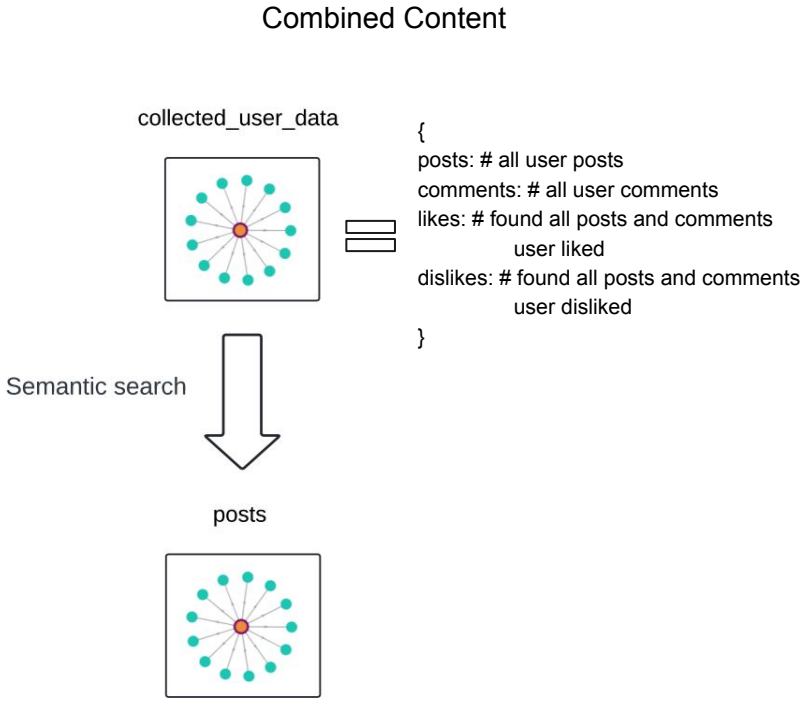
- Qdrant handles vector searches which is key for providing fast and relevant recommendations.
- Python helps us develop quickly and integrates smoothly with Qdrant.
- React improves the user experience with dynamic, interactive interfaces.
- Altogether, this stack ensures a smooth flow of data, boosts performance, and engages users with personalized content delivery.

Algorithm Architecture 01

Stored all data points in a Qdrant vector database



Algorithm Architecture 02



Algorithm Architecture 03

Eliminated ghost users from the suggestion algorithm

Used condensed content to identify ghost users.

{
posts: “”
comments: “”
likes: “”
dislikes: “”
}

Ghost users’ profile information also completely empty database

Algorithm Architecture 04

- Assigned weights to the respective vectors, and combined into one vector
  - combined_embedding = (
    0.4 * posts_vector
    + 0.3 * comments_vector
    + 0.2 * likes_vector
    - 0.1 * dislikes_vector
)
- Used this vector to semantically search posts with Qdrant similarity search feature
  - Pulls top 20 posts most similar to a singular user, ranks by similarity score, can accomplish this for all users
- Used a similar algorithm to recommend users

What's next?

- ➔ Refresh post and user recommendations
- ➔ More analytics for entire user base
- ➔ Add trending posts/topics as recommendations