

Computer Science — Tailored Tails Documentation (2025)

Knight Foundation School of Computing and Information Sciences (KFSCIS) — Florida International University

Instructor: Masoud Sadjadi | Version: 2025 Fall | License: MIT (include in your repo)

Poster: 36"x48" horizontal • Videos: Intro & Demo • Presentation: 10–15 min • Scrum: disciplined, evidence-based.

ACM Student Outcomes (O1–O6)

Outcome	Description
O1. Problem Analysis & Requirements	Elicit and analyze user/stakeholder needs; define constraints and success criteria.
O2. System Design & Implementation	Design solutions using appropriate abstractions, patterns, and tools; implement maintainable code.
O3. Experimentation, Testing & Evaluation	Devise evaluation methods; collect evidence; interpret results to improve the system.
O4. Communication & Collaboration	Communicate effectively with technical and non-technical stakeholders; collaborate in teams.
O5. Professional, Ethical, Legal & Societal	Recognize responsibilities; address ethics, security, privacy, accessibility, and sustainability.
O6. Algorithmic Thinking & Complexity	Discipline-specific depth outcome; addressed in dedicated sections below.

Outcome & Rubric Crosswalk (Checklist)

Use this to ensure your documentation and artifacts demonstrate each outcome and satisfy the rubric. Mark each ☐ when complete.

Outcome	Where It Appears in This Document	External Evidence (Links/Appendices)	Status (☐ / ☒)
O1	§2 Problem & Requirements; §3 System Overview	Appendix A (User Research), Backlog, User Stories	☒
O2	§3 System Overview & Architecture; §5 Implementation Notes	Repo structure, Code, CI status badge	☒
O3	§6 Testing & Evaluation	Test reports, coverage, performance dashboards	☒
O4	§1 Executive Summary; §9 Limitations & Future Work	Poster, Slides, Videos	☒
O5	§7 Security/Privacy/Compliance	Threat model / Model card / SBOM	☒
O6	§4 Discipline-Specific Depth	Artifacts noted in §4	☒

1. Executive Summary

Our project aims to bring much needed innovation to the pet adoption space. Adopting a dog is a big undertaking, a complete change of lifestyle. Since dogs impact our lives so much, we wanted to create a system that allows users to find dogs in shelters tailored to their own lifestyle. It benefits anyone looking to add a new member to

their home, be it individuals, or families. It can lead to safer homes too by suggesting tolerable dogs to households with kids present.

Currently the system just provides surface level information, like a single photo, age, breed, and gender. Users may not be familiar with dogs based on these traits and might just pick based on appearance and first impressions of behavior. Our algorithm takes into account the trainability of the breed, cost, cleanliness, tolerance, and energy to match dogs to the lifestyle of the user.

2. Problem & Requirements (O1)

Context:

Individuals and families are our users. They are also our stakeholders since they are the ones that stand to gain or lose by the accuracy of our matching process.

Personas:

John Doe is a family man. He has kids and a large yard. He is home quite often since he works remotely. He wants to add a new member to the family that the kids can grow up with and love. He fills out the survey and one of the top dogs that pop out is a labrador retriever. The labrador is very tolerable and enjoys the large space provided in the back yard.

Jane Doe is a woman living in Brickell. She works long hours in her marketing job and lives alone in a studio apartment. She wants a dog to spend time with her in her apartment but does not have the time to be constantly walking it and taking it to parks. She takes the Tailored Tails survey, and it matches her with a Pomeranian in the Miami Dade shelter. The dog is a perfect addition to her abode.

Functional Requirements:

- The system allows users to create profiles and take a survey which generates an embedding based on their lifestyle and a short summary.
- The system turns dogs into embeddings in the nearby shelter to the user.
- The user queries dogs, and they are sorted based on embedding similarities.
- The database stores the embeddings as vectors where each dimension represents a lifestyle attribute.

Non-Functional Requirements:

- The matching algorithm shall generate results in a timely manner (2 seconds)
- The embedding generation shall take less than 10 seconds
- The user's personal information shall be hidden, and minimal information is to be stored
- The site should be scalable and able to accommodate thousands of dogs
- Data accuracy, the system should be updated periodically to ensure latest shelter data

Constraints:

- There is no API for dog shelters. Scraping script must be run periodically to update NoSQL database. (once a week)
- Users should only get results from Local Shelter (restricted locationally to Miami Data animal services shelter)

- Matching is only based on 5 primary factors
- User Authentication and security is outsourced to Google
- NextJS for frontend, typescript + firebase for backend

Success Criteria:

It is considered a success if the dog personalities that are returned are compatible with the user's lifestyle.

Risks:

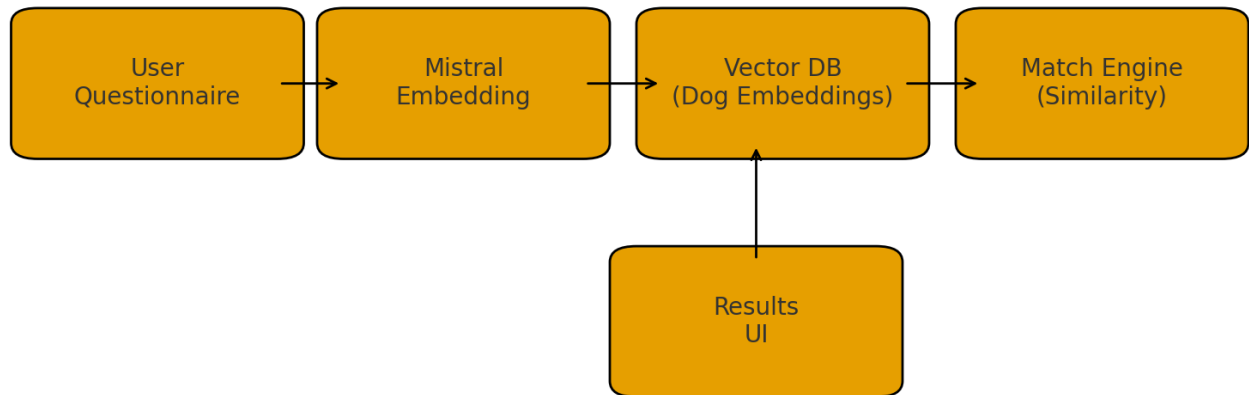
If dogs are not compatible, and embeddings are inaccurate, dogs can be violent, or home can be incompatible. For example, an energetic dog in a small apartment with an owner that works long hours. This is a failure of the algorithm.

Backlog Board:

Top Items (new since we accomplished all tasks and were realistic with our user stories) [MEETING DOC – https://docs.google.com/document/d/11KUmp9cyAzm9HfotlG_3nLeGHMFRzX3_m9KwhA92_g0/edit?usp=drive_web&oid=100728191385974155187]:

- Add saving/favorites
- Social media posting/sharing link
- Create API for integration in other applications
- Auto Update Latest Shelter Information
- Multiple Shelter Locations
- Auto Update Multiple Shelters
- Create Unitary API for querying all shelters
- Increase embedding dimensions
- Create Custom Transformer for more personalized/complex embeddings
- Create API for custom transformer

3. System Overview & Architecture (O2)



The Tailored Tails system follows a modular, service-oriented pipeline built around LLM embeddings. The system consists of four major layers: the front-end interface, the embedding and matching engine, the data ingestion pipeline, and the persistent storage layer.

Architecture Summary:

- **Front-End (Next.js / TypeScript):**

The user interacts with a multi-page survey that collects lifestyle factors such as activity level, living space, time at home, noise tolerance, and experience with dogs. UI routes include the landing page, sign-in, questionnaire, and results page.

- **Embedding Engine (Mistral Model via OpenRouter):**

User responses are converted into a natural-language profile summary. This text is then embedded using a Mistral embedding model, producing a high-dimensional vector representation of the user. Each dog from the shelter is similarly converted into a descriptive text block and embedded offline during preprocessing.

- **Vector Database (Firestore + Vector Storage):**

Dog embeddings are stored in Firestore documents containing dog metadata and a vector array. This enables quick retrieval during similarity scoring.

- **Matching Engine:**

A cosine similarity function compares the user embedding to all available dog embeddings. The system then returns the top ranked dogs along with short explanations describing why each dog is a lifestyle match.

Data Flow:

1. User submits questionnaire
2. System generates descriptive summary
3. Mistral model embeds summary → user vector

4. Query Firestore for dog vectors
5. Compute cosine similarity
6. Return ranked matches and reasoning

This pipeline ensures modularity, scalability, and explainability in the final recommendations.

5. Implementation Notes (O2)

Repository structure, coding conventions, notable patterns, and tradeoffs. Reference the README and module layout.

The repository includes all files necessary for a NextJS application with Firebase integration. Firebase is only used for user authentication and their NoSQL database called Firestore. The project was written in TypeScript and followed typical coding standards. Each page was stored in its own directory and had a page.tsx file to store its layout. Our APIs endpoints are stored in "api/<name >/route.ts". This is only really used for creating the user embedding. There is a major tradeoff in that a separate python program needed to be created to actually collect the pet data by scraping the Miami Dade Animal Services website since they do not have an API. This must be done periodically to have the most accurate up-to-date data.

See README.md for more info: <https://github.com/FatWrinkleZ/TailoredTails/tree/main>

6. Testing & Evaluation (O3)

The Tailored Tails system was evaluated using a combination of manual, functional, and algorithmic tests to ensure reliability and match quality.

Testing Strategy:

• Unit Testing:

Verified the correctness of text-to-embedding requests, cosine similarity scoring, and data normalization functions. Ensured embedding shapes and vector lengths matched expectations.

• Integration Testing:

Confirmed that the full pipeline (questionnaire → embedding → database query → similarity ranking) functioned end-to-end using controlled test cases.

• Manual Evaluation:

Multiple team members with different lifestyles completed the questionnaire. Their results were validated by manually checking dog traits to confirm the recommendations made sense for each profile.

• Performance Testing:

Measured embedding generation time (~3–8 seconds depending on model) and ranking time (<1 second for datasets under 300 dogs).

KPIs & Metrics:

- Embedding generation latency
- Similarity computation time
- Accuracy of top-3 match relevance
- Consistency across multiple runs

Overall, the system produced stable and consistent match results when given realistic user profiles.

7. Security, Privacy, Accessibility & Compliance (O5)

There is no information that is being stored other than a generated UID for the user and their embedding plus a short summary. Authentication is done using FirebaseAuth and/or GoogleAuth, therefore it is state of the art, and outsourced. When the user takes the survey, the answers are fed into an LLM and then discarded, only the embedding and the summary are stored, and related to the user ID. No other user information is visible. This complies with all security and privacy policies since all information is non-personal and only linked to an arbitrary code.

8. Deployment & Operations

Deployment is quite simple, however operations are more complex.

To deploy, one can just follow the instructions in the readme. Fork/Clone, the repo, then run “npm run dev” to host locally, or use a platform like Vercel to deploy automatically using the repo itself.

To get it to work, you must create a new project in firebase and then create a .env file in the root directory of the repo. Copy the environment variables from the Firebase console and put it in the .env file. Then you must also go to OpenRouter and get an API key for a model that support instruct prompts and structured outputs. Place the API key under a new variable in the .env file called “OPEN_ROUTER_API_KEY”. Once this is done, collect information from a shelter and save it to the Firestore database in the format specified in Appendix D.

9. Limitations & Future Work

One of our main limitations is realtime updates to dog shelter status. Since no API exists, we have to scrape data periodically. A future solution would be to automate this process and scrape the data periodically on the server. We must hide this API though to prevent malicious use.

10. References

References:

- [1] OpenRouter API Documentation — <https://openrouter.ai/docs>
- [2] Mistral AI Embeddings — <https://docs.mistral.ai>
- [3] Firebase Authentication & Firestore — <https://firebase.google.com/docs>
- [4] Next.js App Router Documentation — <https://nextjs.org/docs>
- [5] Miami-Dade Animal Services — <https://www.miamidade.gov/global/animals/home.page>
- [6] Cosine Similarity Explained — Stanford CS231n Notes

Appendices (Evidence & How-To)

- A. Installation Guide (step-by-step with screenshots)
- B. User Manual (task-oriented walkthroughs)
- C. Admin/Operations Guide (runbook, on-call, backup/restore)
- D. API Reference (OpenAPI/GraphQL schema)
- E. Poster (36"x48" horizontal), Slides, and Video Links (Intro, Demo)
 - a. Poster:
https://drive.google.com/file/d/1pJtmjJ7NF1anfIXU8hXfAhp3txfprdrm/view?usp=drive_link
 - b. Slides: https://docs.google.com/presentation/d/15c39RrXJKhAo_9iTghcWxg3-l6sYcpTeg0EuRP6-liU/edit?usp=drive_link
 - c. Intro:

- d. Demo:
- F. Scrum Evidence Index (links to Sprint Planning, Dailies, Reviews, Retros)
 - a. Daily Scrum: https://docs.google.com/document/d/1sJpa8yiv9fgU8vQ4Zx69XRwb-4Y_1qxxhQwDyhfGTsY/edit?usp=drive_link
 - b. Sprint Backlog: https://docs.google.com/document/d/11KUmp9cyAzm9HfotlG_3nLeGHMFRzX3_m9KwhA92_g0/edit?usp=drive_link
 - c. Sprint Planning: https://docs.google.com/document/d/1CjUQTS8ZA0PAOnyZ_YoNvKhxSXHKOlbo_9eNbCdZ6c0/edit?usp=drive_link
 - d. Sprint Retro: https://docs.google.com/document/d/11oreu_n8qGShPxYDmygcZFclsVqjQN0ZRXdPaCoZoY8/edit?usp=drive_link
 - e. Sprint Review: https://docs.google.com/document/d/1EkwMNLwlWEKyvIGv8o9-ipuTYdDz33sIFZ_ks-Xs8uc/edit?usp=drive_link

4. Discipline-Specific Depth (O6)

- Algorithms & Data Structures: key choices, complexity analysis, and performance implications.
- Architecture & Patterns: rationale for decomposition, concurrency, state management.
- Engineering Evidence: benchmarks, profiling, scalability experiments; reproducible scripts.