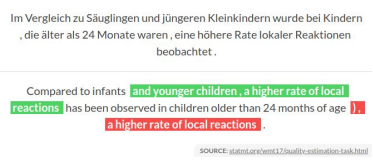
Modular Annotation for Natural Language Processing

Diego Montes | Bengisu Cuneyit | Jisoo Kim |
Shreyas Chickerur | Sahithi Tummala | Saimonish Tunguturu

Data Annotation for Contemporary ML Challenges

Inspiration

1. Data is the backbone of machine learning models, yet real-world data is messy: data annotations format this data in a way that a model can learn from.
2. In the past, we had used and attempted to extend an open-source annotation tool: cdQA-annotator; however, there were a number of pitfalls:
   a. No centralized storage: annotators had to manually download annotations after each annotation session and upload them to Sharepoint.
   b. Difficult to implement new features: the tool was no longer maintained, and the code base's infrastructure did not support pluggable features.
   c. A lack of a project structure and workflow.
3. As such, our project's specifications revolved around these three missing features.

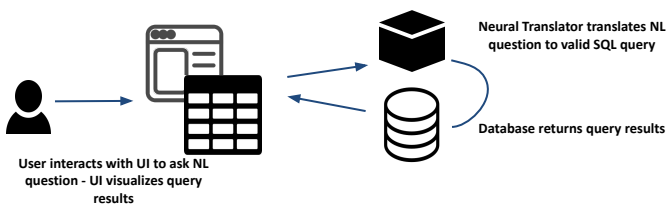


An Example Translation Data Annotation:

- The top portion displays the text to-be-translated.
- The bottom portion shows the translated text, annotated for errors.

Why Another Annotation Tool?

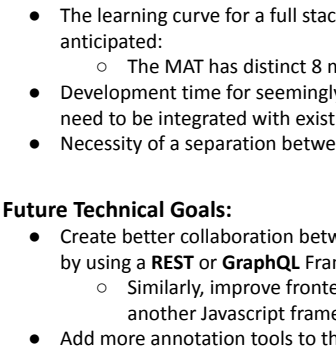
1. Distinct from many open-source annotators, a priority was making our annotator pluggable for future tools and software contributors.
2. Similarly, despite a number of paid annotation tool services that offer the features we were looking for, we wanted to create a tool that could annotate data for the newest ML/AI challenges:
   a. One such example is NL2SQL: no annotation service offers a tool for natural language to SQL annotations (depicted in the diagram below)



Annotating Workflow

Project Management

1. Organization of data annotations becomes increasingly important as annotation projects get larger, this calls for:
   a. Store and retrieve functionality for data and annotations.
   b. Project and task structure to distribute annotation workload.
   c. Workflows to push annotations to other storages automatically.



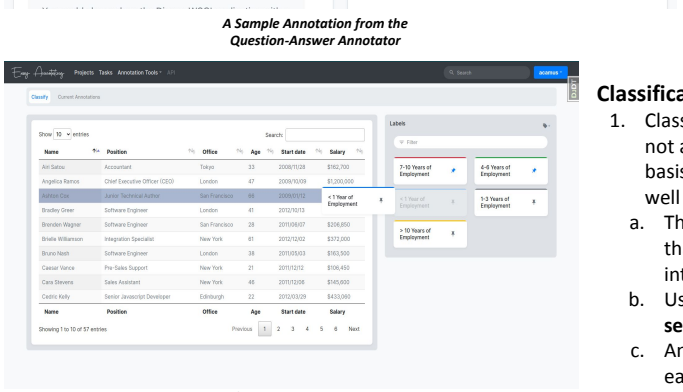
Annotation Pipeline

1. A user creates a project, inviting other members if necessary.
2. Any team member uploads data belonging to the supported formats.
   a. Tasks are automatically created and divided among team members.
3. As annotations are created, project statistics are updated and at any point all annotations for a project can be downloaded.
4. Optionally, annotations can be periodically sent to an AWS or GCP storage.

Question-Answer Systems, Classification, and SQL Annotators

Generic Annotation Workflow

1. The end-user uploads a supported file type (**.db, .csv, .txt, .md**) directly to the annotator or pulls a previously uploaded file from a **project.**
2. The user creates an annotation, specific to the annotator being used:
   a. For QAS, the user writes a **natural-language question** and selects an answer span.
   b. For SQL Annotations, the user writes a NL question and a corresponding SQL query.
   c. For Classification, the user drags labels onto table rows, classifying them into categories.
3. The annotation is sent to **Django** for additional validation and bleaching.
4. The annotation is saved to **Postgres** and project statistics are updated .

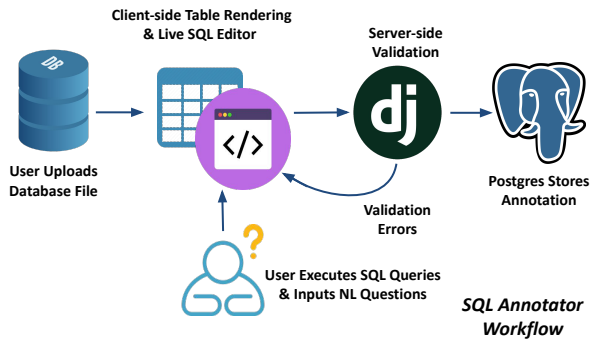


A Sample Annotation from the Question-Answer Annotator

Classification Annotator Table View

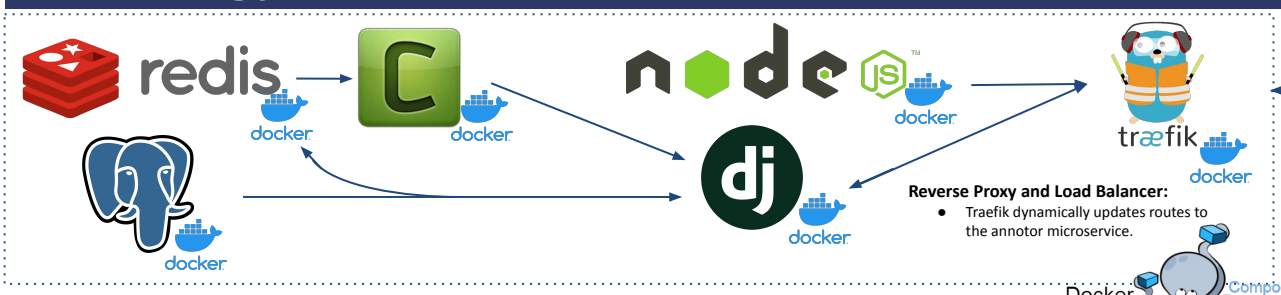
Classification Tool Motivation

1. Classification of natural language (NL) is not always on a document-by-document basis: text data can be found in tables as well
   a. The tool should include an annotator that can **classify rows of tabular data** into a distinct categorical column.
   b. Users should be able to **make and search for class labels.**
   c. Annotating a given row should be as easy as possible.



SQL Annotator Workflow

Technology Stack



Reverse Proxy and Load Balancer:

- Traefik dynamically updates routes to the annotor microservice.

Frontend:

- A combination of Bootstrap4, Vanilla Javascript, Jquery, and various other libraries create an interactive UI.

Database & Cache:

- PostgreSQL stores annotations, projects, tasks, and workflow data.
- Redis caches frequently visited projects to allow for faster access.

Background Tasks:

- Celery workers update project statistics and process uploaded data without causing longer page-loading time.

Backend:

- Django process incoming requests by querying the database and running view-specific logic.
- Node.js builds frontend dependencies, including compression and concatenation.

Containerization:

- Docker Compose runs all the Docker containers locally.
- K8's manages the scaling and deployment of these containers in production.

Conclusions & Future Goals

Conclusions:

- The learning curve for a full stack of technologies was larger than initially anticipated:
  - The MAT has distinct 8 microservices in its stack.
- Development time for seemingly small features increases drastically as they need to be integrated with existing features.
- Necessity of a separation between frontend and backend.

Future Technical Goals:

- Create better collaboration between the frontend design and the backend logic by using a **REST** or **GraphQL** Framework.
  - Similarly, improve frontend design development by using **React** or another Javascript framework versus **Vanilla Javascript.**
- Add more annotation tools to the MAT suite
  - **Multiple choice image labeling**
  - **Audio transcriptions**
  - **Named entity labeling**
- Implement integrations with other data services, such as **Pure Storage.**

References & Acknowledgments

References:

- Django (Version 1.5) [Computer Software]. (2013). Retrieved from https://djangoproject.com.
- Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. Linux Journal, 2014(239), 2.
- Bootstrap (Version 4.6) [Computer Software]. (2021). Retrieved from https://getbootstrap.com/

Acknowledgments:

- We would like to thank our:
  - Corporate Partner Mentors: Sarah Rodenbeck and Justin Gould
  - Student Mentor: Rishabh Rajesh
  - And the entire Ford team for giving us this opportunity, letting us access their internal data and systems, and guiding us as we created and refined this project.
- We would also like to thank: Dr. Mark Daniel Ward, Ellen Gundlach, and Maggie Ann Betz for supporting us during the Academic Year and providing us with the resources for the successful completion of our project.