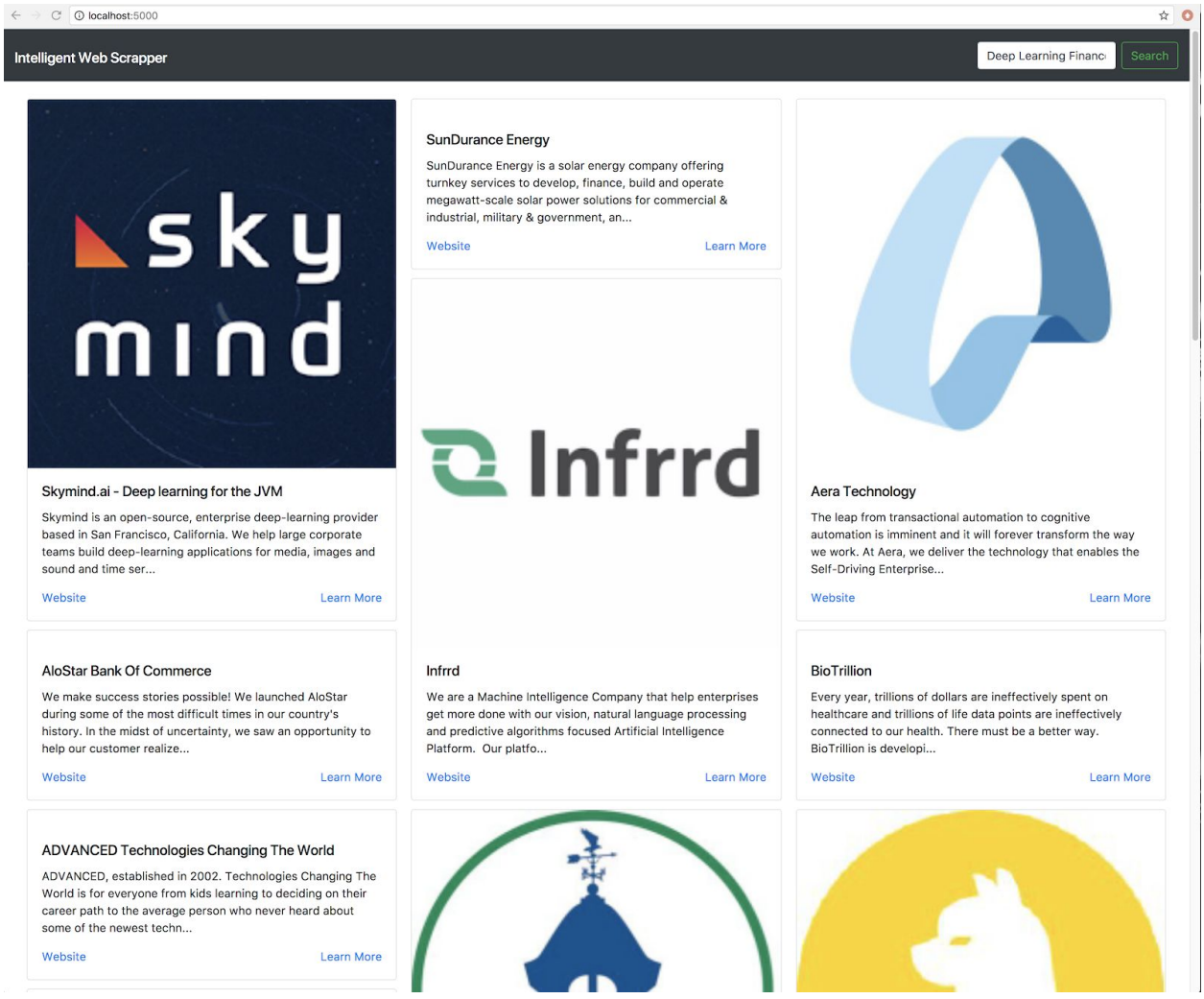
Intelligent Web Crawler

Open Ecosystem Network

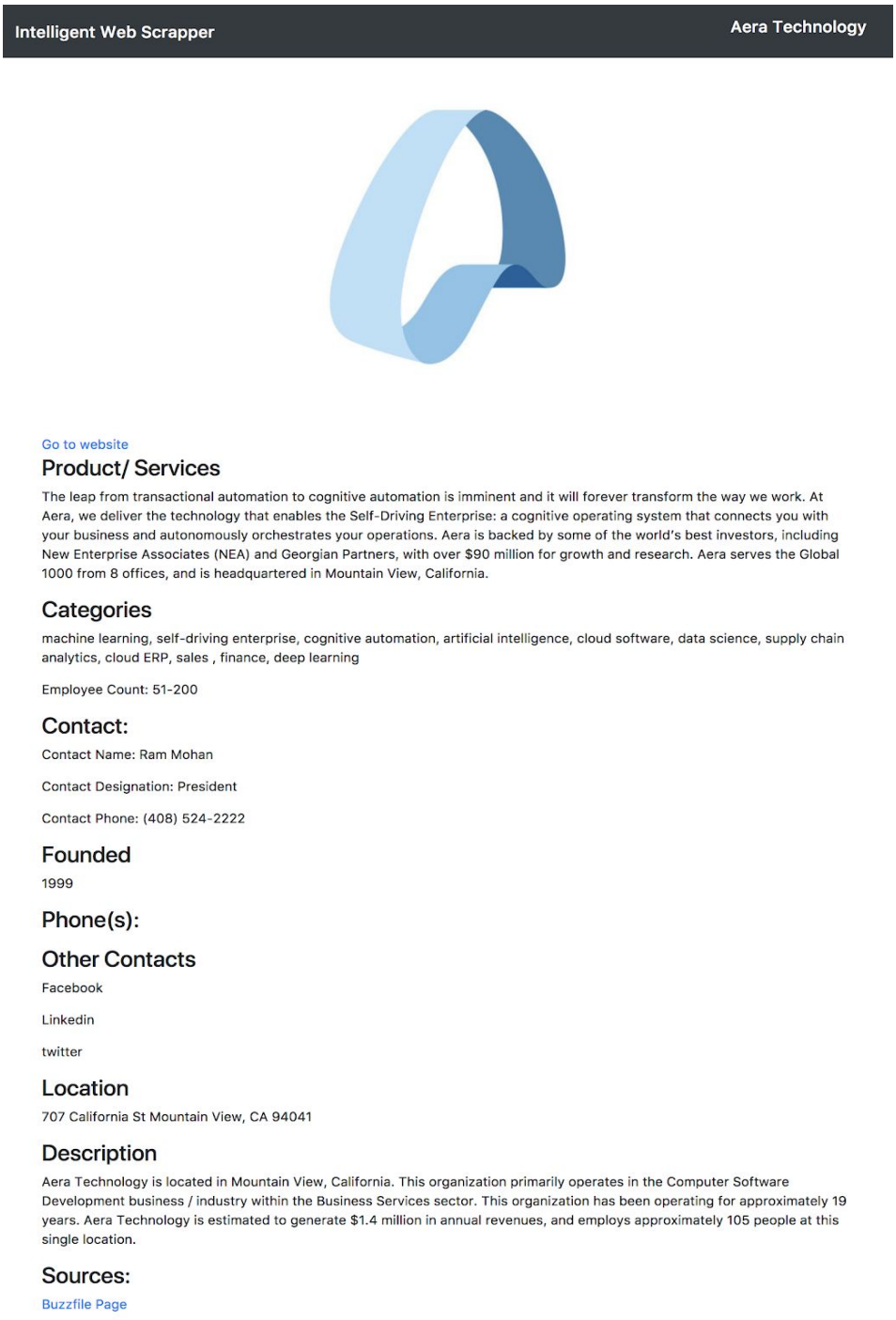
Summary

The web scraper uses a combination of LinkedIn, Bing API and a general web scraping algorithm to generate a list of companies related to specific interest tags. Bing API was used instead of google because there is no google API equivalent to their standard search. The results are displayed using a custom algorithm to put companies Open Ecosystem Network would likely target (startups that are rapidly growing) first. The program uses a simple web interface to search and navigate the responses.

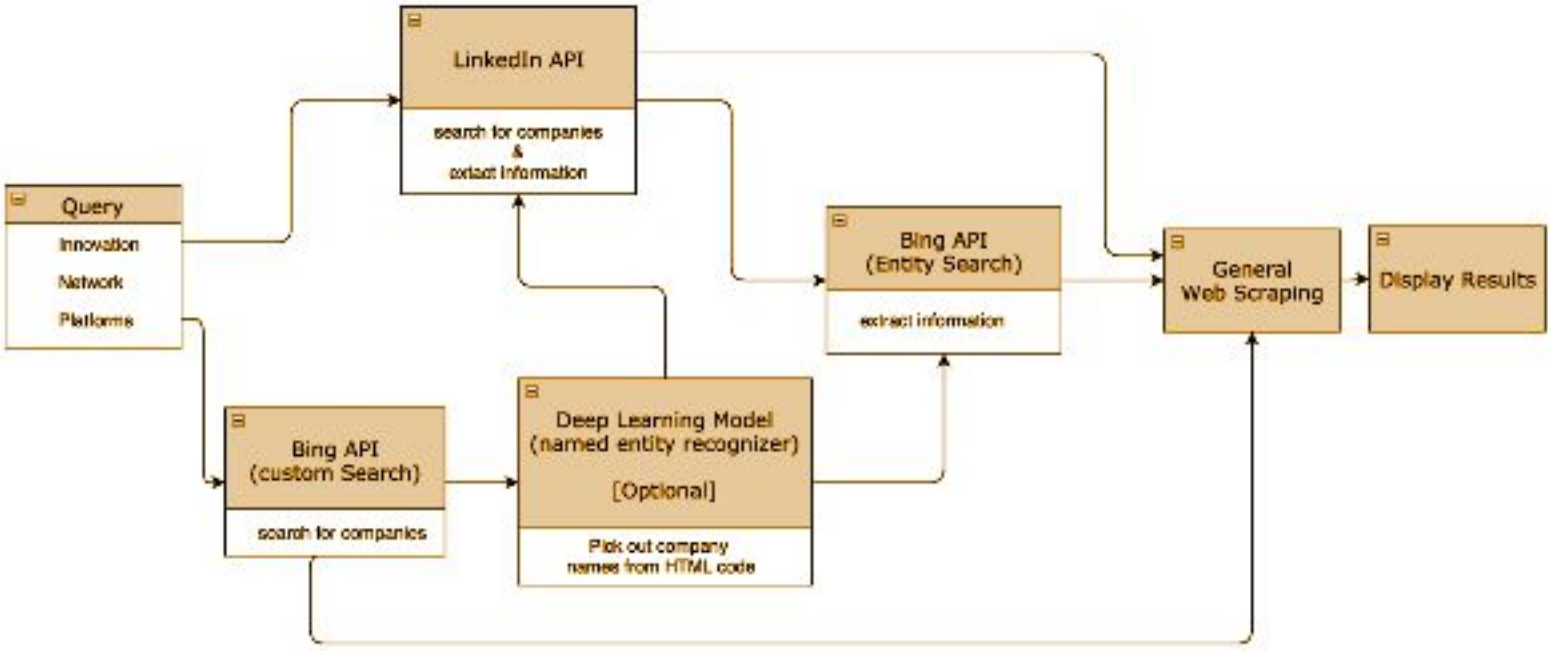


Detailed Company Description

Below is an example of what a detailed company description would look like. It takes about 3-5 seconds for each individual company page to load after clicking on the icon from the main search results page. The program performs several searches and scraping for the necessary information after the user clicks on it.



We can reduce the speed significantly with more time by adding asynchronous request and implementing a database. This could be easily implemented if the customer desires.



1. Query
    - The user enters the desired query into the search bar
2. Bing Custom Search
    - The program queries the Microsoft database, navigates to each of the response urls and grabs all the text
3. Deep Learning Model for Named-entity recognition [Optional]
    - Input: text from the Bing search query
    - Output: The model recognizes named entities (e.g. companies, people, locations) from the text.
    - This part of the program is not enabled by default. The usage of this component will increase the response time of the query to ~30 seconds compared to almost instantaneous.
4. LinkedIn API
    - The LinkedIn API uses the query and returns a list of companies
    - The list of companies is sorted based on relevance to the keywords, company size, and operational status
    - Information about the companies is also extracted with the API
5. Bing Entity Search
    - Bing Entity grabs information that might not have been available on LinkedIn and for companies that don't have LinkedIn
6. General Web Scraper
    - A generalized web scraping algorithm is applied to company websites to find missing criteria. It collects phone numbers, emails, and people(leverages the NER model)
    - The program searches BuzzFile.com for general information, as well as the the company's website

7. Display Results
    - The results from the query are displayed in the web browser in order from most to least relevant

Deep Learning Model

Why use deep learning? To filter out unnecessary parts and find what we're looking for. The deep learning model is used for named entity recognition(NER) which "labels sequences of words which are the names of things, such as a person and company names." It leverages anaGO, an open source Keras implementation to perform the NER.

[Assumes you've already installed python3.6 and the pip packages]
TO TRAIN: You'll need to DOWNLOAD the GloVe embeddings here and unzip into the `data/` directory in the root of project. Also download the conll2003 dataset here and unzip into the 'data/' directory as well.

Performance

The model is originally trained on the CONLL dataset and achieves a F1 score of ~92%. However, the accuracy doesn't translate to web text so the model needs to be fine-tuned. After building a custom dataset and training on it, the accuracy of the model didn't improve greatly. The model was trained using transfer learning from a model trained using GloVe embeddings and trained on the CoNLL dataset, as well as training exclusively on a personal dataset.

Getting The Base Model

The base model is included in the <PROJECT_ROOT>/srcs/models/base_model directory and will be called by 'ner.py'. (See below on ways to interact with the model)

TRAINING THE BASE MODEL: After you've installed the GloVe embeddings and conll 2003 dataset(above) then run `python ner.py train_base_model`. You can use the `--help` flag to see options and command information. You can monitor the training in Tensorboard. After the training is complete, a keras model will be saved to 'data/base_model/`.

Build A Custom Dataset

There's a search and label CLI that'll take a query keyword, compile text by crawling relevant links, then it'll ask the user to label each word. What text is being fed into the model? Each link is parsed and the title, headers, and paragraphs are compiled. Afterwards each word in the corpus is labelled as either an organization, person, location, miscellaneous, or to be ignored.

Each query's words and labels will be saved to a separate text file in the 'data/queries/' directory.

Train On The Dataset

After you've ran the search and labeller, you'll notice that that are many txt files in the data/queries directory but you'll need a lot of training examples to have a decent model. To train the model run `python ner.py train` and you'll begin training on the data you've gathered.

A model will be saved to the 'data/custom_model' directory.

Run the Model

Execute `python ner.py test_model <SENTENCE> --model_dir=<MODEL_PATH>` to see test output.

What's being used in the web scraper pipeline?

'run_model()' in ner.py is being called to 'scrape' a domain. The website's text is retrieved and used as input to the model. The output will be:

```
>>>
{
   "words": [
      "President",
      "Obama",
      "is",
      "speaking",
      "at",
      "the",
      "White",
      "House."
   ],
   "entities": [
      {
         "beginOffset": 1,
         "endOffset": 2,
         "score": 1,
         "text": "Obama",
         "type": "PER"
      },
      {
         "beginOffset": 6,
         "endOffset": 8,
         "score": 1,
         "text": "White House.",
         "type": "ORG"
      }
   ]
}
```

Relevant Files:

srcs/ner.py *- NER for web parsing*

- *`evaluate` : test a models performance on a formatted TSV file*
- *`predict` : performs NER on a text file*
- *`test_model`: performs NER on a sentence. Can be used to test the model*
- *`train`: trains the model on a custom dataset (can use weights from base model or random weights)*
- *'train_base_model': trains the model on the conll dataset*

srcs/search_and_label.py *- Google search query for building a dataset and label CLI*

>>> python search_and_label.py
>>> What do you want to search? : <QUERY>
>>>
>>> LINKS:
>>> https://related_search_query_link.html
>>> https://related_search_query_link.html
>>> https://related_search_query_link.html
>>> Found headers NUM
>>>
>>> Data Labeling
>>>
>>> 'This is a sentence.`
>>> Enter the label for `This` []: <LABEL>
>>> Enter the label for `is` []: <LABEL>
>>> Enter the label for `a` []: <LABEL>
>>> Enter the label for `sentence` []: <LABEL>
>>> Enter the label for `.` []: <LABEL>
>>> ....

Output will be written to a tab-separated value text file.

How to improve the model?

The model needs more data to get better at recognizing named entities. Adding more diverse datasets from around the web to extract information would increase the model accuracy. Ideally, the model takes in input keywords and spits out entities that are related. Afterwards the end user can decide which entities are most relevant.