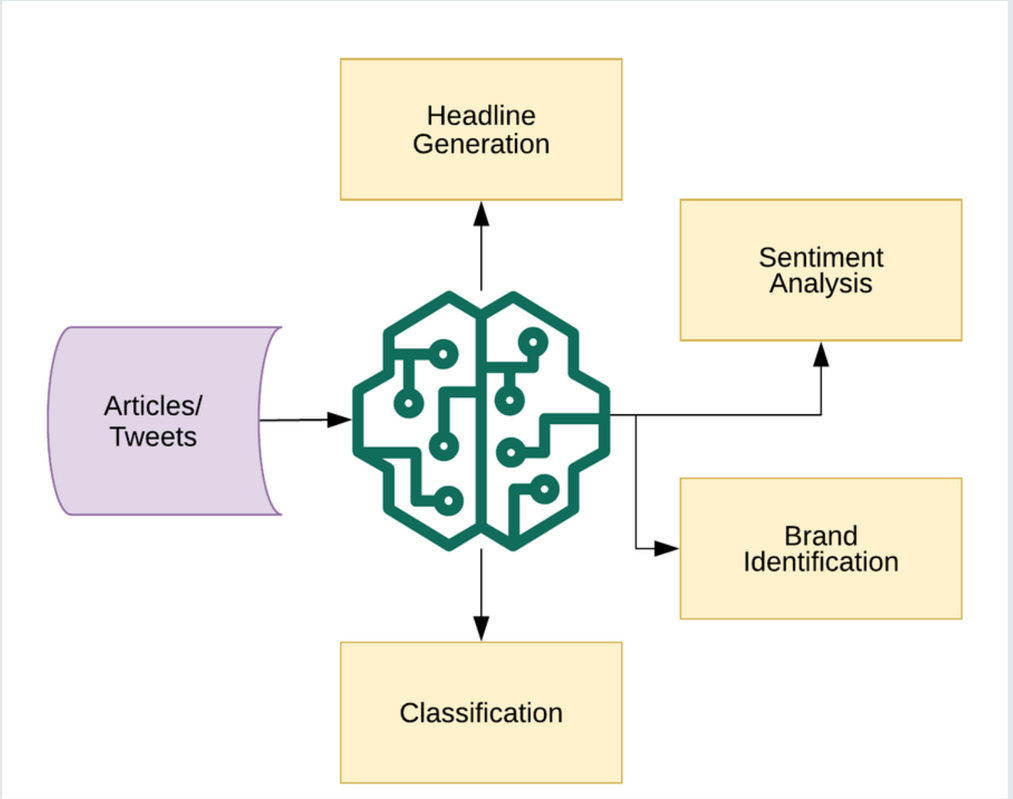
BRIDGEi2i's Automated Headline & Sentiment Generator

Automated identification, summarization, and entity-based sentiment analysis of mobile technology articles and tweets

Table of Contents

The Problem

Building an **automated system** for **segregation** of text into **mobile tech** and non-mobile tech classes and further **identifying** the **mobile brands** associated with the mobile tech articles/tweets and **finding** the associated **sentiment** with the brand. Further building an automated **headline generator** for the mobile tech articles.



The Dataset

The Tweets

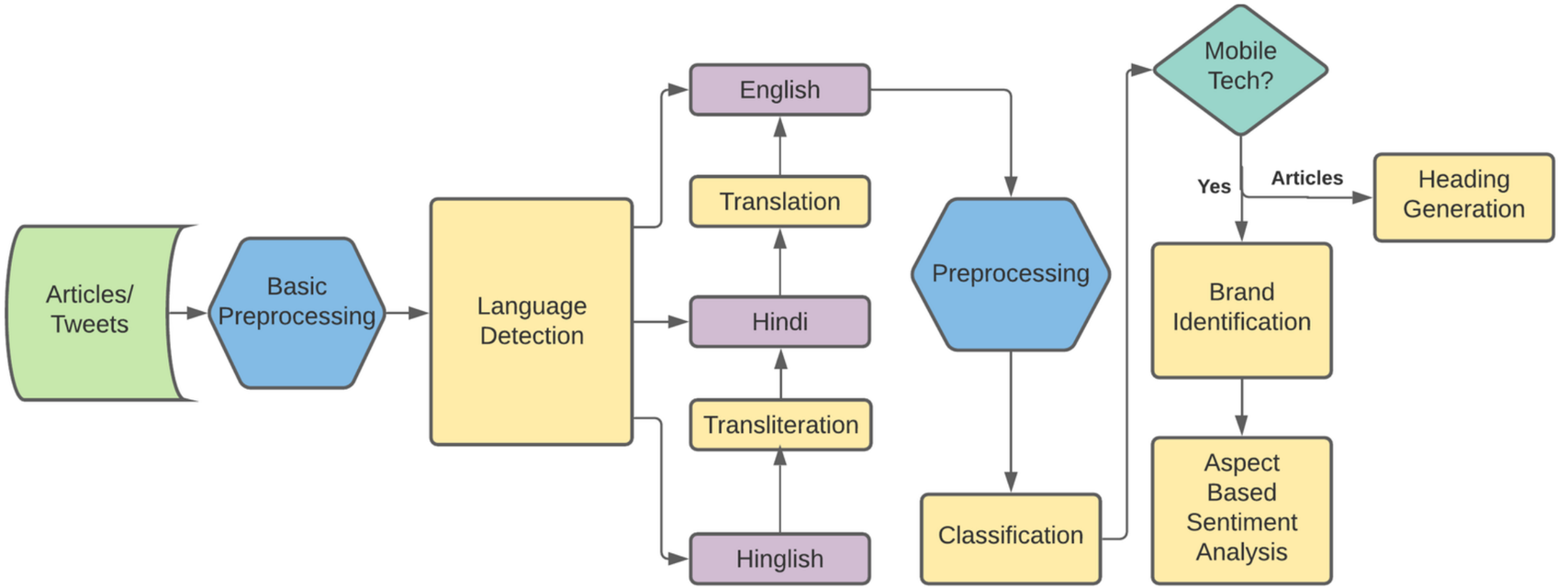
4000 tweets in English, Hindi, and Hinglish languages, with 1000 belonging to mobile tech and 3000 belonging to non-mobile tech. The data had a lot of ***duplication***, and most non-mobile tech tweets were ***politically opinionated***.

The Articles

4000 articles in English, Hindi, and Hinglish languages, with 1000 belonging to mobile tech and 3000 belonging to non-mobile tech. The data had a lot of ***duplication***, and most non-mobile tech tweets were ***news articles***.

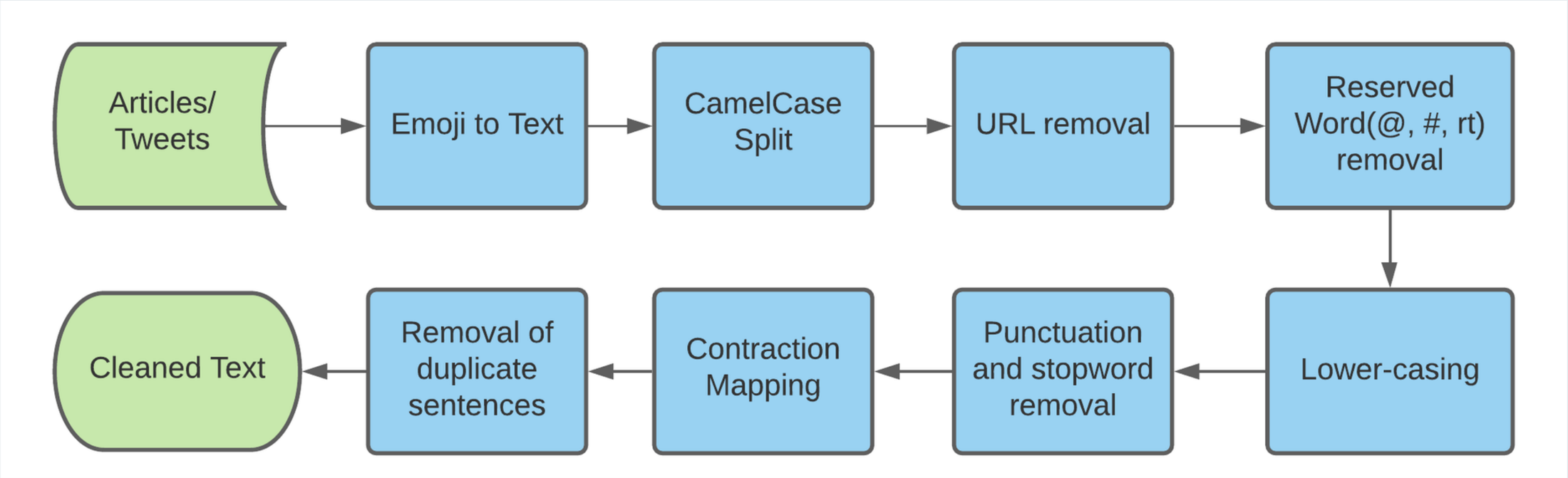
Our Approach

We developed an ***end-to-end***, ***scalable***, and ***novel pipeline*** for the identification, summarization, and entity-based sentiment analysis of mobile technology articles and tweets using various deep learning architectures for language modeling. Our **main focus** was on using **lightweight algorithms** without compromising on the **quality** of **predictions**.



Preprocessing Pipeline

We carefully crafted a robust and fast data preprocessing and cleaning pipeline to ensure that we can remove the maximum amount of noise with the least amount of information loss from the given dataset.

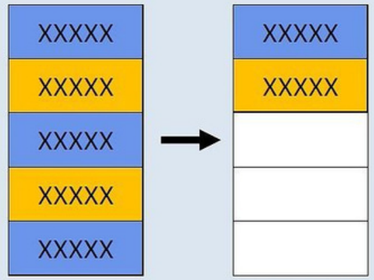


Language Agnostic Text Cleaning

Emoji to emoji sentiment conversion using ***demoji*** package

Removing URLs, RTs, QTs, and hyperlinks. Removing @, #, and _ in the user handles

Pictographs, Alchemical symbols, Geometric Shapes, Supplemental Arrows, etc. were removed from the text.



Remove Duplicates

Why do we care?

If a train test split is performed without removing the duplicates, there will be a ***heavy data leakage*** in the validation pipeline leading to an inaccurate validation set

Duplicates Handling

- For the removal of duplicate sentences, we developed a **unique graph-based clustering** algorithm using ***Levenshtein distance***, since there were a lot of examples that were different because of the presence of gibberish or very uncommon words. We were unable to identify them just by using existing duplicate methods in pandas.

- So we developed a clustering algorithm that clusters similar words with Levenshtein distance less than a certain threshold and then one data point from each cluster is selected as a unique example.

- On removing the duplicates, we noticed that only **40%** of the dataset had a unique text.

Language Detection

Detecting Pure Hindi Text

The Hindi characters (Devanagari script) have hexadecimal characters between **0x0900** and **0x097F**. Thus any word written in Hindi can be identified just by checking its Unicode.

Detecting Hinglish Text

We built a **character** level Bidirectional **LSTM** based model which was trained on a self-made dataset that contained **Hinglish** and **English** words. It was able to identify **Hinglish** text with up to **95%** accuracy.

Transliterating Hinglish Text

Strategy Applied

To deal with **code-mixed** language, we **transliterated** the sentences to their intended language so that the code-mixed language was converted into a pure-Indic language.

Datasets used

We collected data from the following sources: *Xlit-Crowd*, *NeuralCharTransliteration*, *Xlit-IITB-Par*, *BrahmiNet Corpus*, *Dakshina dataset*, and Hindi word transliteration pairs.

Model Applied

We made a ***4 layered Vanilla Transformer***. The model gave an overall BLEU score on the dataset as **50.4** while Google Transliterate API gave a score of **53** with our model being ***2.2x faster*** than Google API.

Translating Hindi

For translation, we used *MarianMT* because it is built on C++, hence is very fast and has 1000+ models therefore is quite scalable. Additionally, MarianMT is used as a *back-bone by Microsoft Bing Translate*. MarianMT is optimized to *run multi-GPU* support as well.

Scalability Component

Converting everything to English made our pipeline more scalable and robust.

Obstacles we faced

Model Architecture

Initially, we wanted to make a ***Linear Attention Transformer*** that would be considerably faster than the vanilla transformer. We tried out the *Performer and Linformer* architectures but we couldn't properly tune them to give good accuracy.

Noisy Datasets

The datasets were ***not very clean***. Some of them were parallel sentences while others had a mix of pure English words in them. A huge part of the cleaning effort required ***manual checking*** to see what we wanted in our final dataset and what we didn't. The process was very time-consuming.

Preprocessing the English Text

Cleaning Contractions

Common **contractions** like *can't(=cannot)* are **mapped** to their **expanded** forms using a **dictionary**.

British English to American English

In addition to that, to bring **uniformity** between **British** and **American English**, we map all words in British English to their American English counterparts.

Cleaning punctuations

Variations in **punctuation** are corrected. We create a dictionary that maps erroneous punctuations to their correct one. Eg: `, ‘, are all mapped to '.

Stopword Removal

Get rid of **stopwords** like a, an, the aka removal of stop words. The list of stopwords is readily available in *NLTK*. We do not apply stemming, since stemming is random cutting.

Lemmatization

Converting word to its base form using a *lemmatizer* from *wordnet* makes our vocabulary less noisy.

Future Prospects

Making Transformers Cheaper

We can shift our transliteration model to faster architecture like *Performer and Linformers* for greater speed.

Multilingual Support

Furthermore, we can train a *multi-lingual transliteration and translation* model that can do transliteration and translation for many languages.

Mobile Tech Classification

The aim of this module is to ***automate the task of filtering*** out the articles and tweets which belong to the mobile tech category so that it can be used for further tasks and analysis.

The Dataset

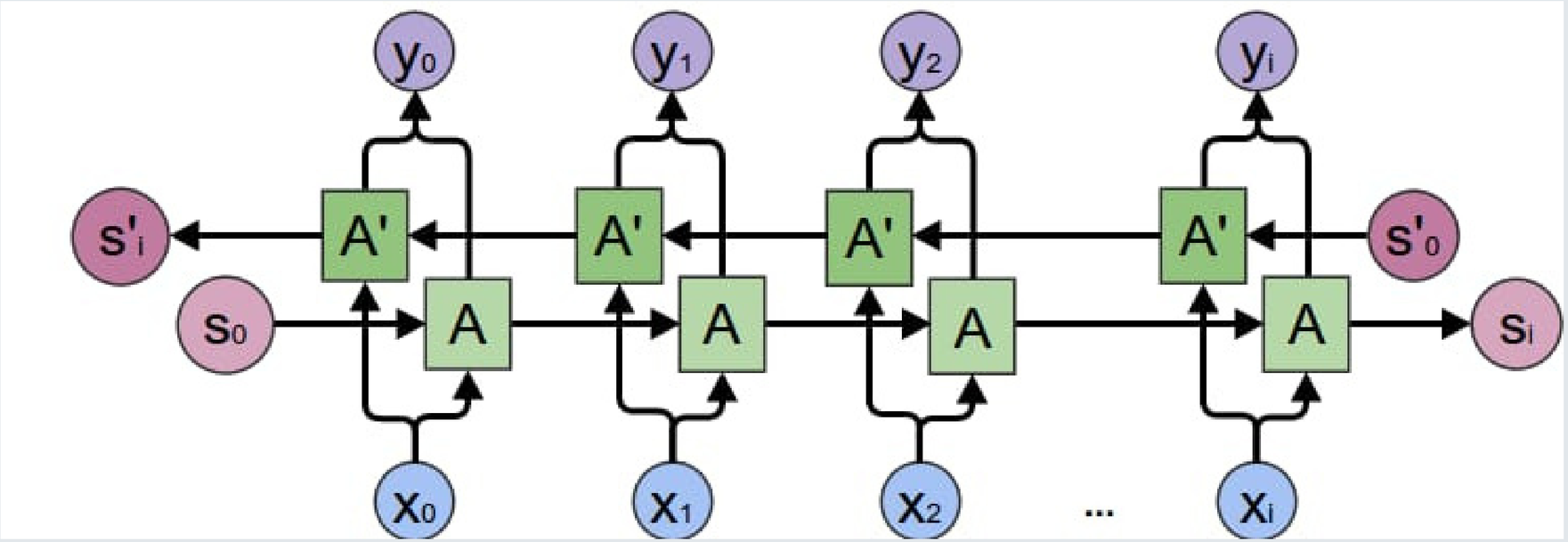
We scraped almost **3000 tweets** related to mobile tech using ***tweepy***(with different brands given as search query) the similar data for roughly 600 articles were scraped from https://gadgets.ndtv.com/mobiles, for the nonmobile tech classification we used 2 publicly available datasets from Kaggle, one being ***demonetization tweets*** and other being a news article dataset.

Validation Strategy

Validation Strategy: New dataset from the scraped dataset was created by rebalancing the classes according to the dataset provided by the organizers. 80% of the given dataset (using stratified split) was added to this new dataset. The remaining 20% of the dataset was used as the main cross-validation set. Further, we also made secondary cross-validation set from 10% of the new training dataset.

Machine Learning Model

We trained 2 separate, *2 layered Bi-LSTM with 100-d Glove Embeddings*, for the mobile tech classification of articles and tweets.



Results

Training F1
0.968

Primary Validation F1
0.942

Secondary Validation F1
0.959

Testing F1
0.92

How did other models do?

Naive Bayes with BOW
0.23

SVM with BOW
0.35

Dictionary search
0.62

Vanilla Bi-RNN
0.91

BERT(base)
0.94

BERT(large)
0.94

Future Prospects

Diversifying approach

We can extract features from the text and use them with *glove encoding* in the top layer of the *Bi-LSTM*.

Collecting better data

Collecting more data that matches the given data distribution so that we can train with a more appropriate distribution.

Brand Identification

For all articles and tweets where the classified theme is ‘*mobile_tech*,’ you would need to identify the brand name.

Our approach

For brand identification, the best method we found is a simplistic ***dictionary-based search approach using regex***. We plan to regularly automatically update our brand name dictionary, this is a ***simple and super-light approach that is also reliable***. We also included common spelling mistakes of the brands e.g. Xiaomi as Xioami, Huawei as Hauwei.

Results

Validation Accuracy	0.97
Test Accuracy	0.94

Other things we tried

We translate all our text to English in our preprocessing module and tried a ***ner-based model (We used spacy's NER model)*** to identify brands, but our model performed better.

One other thing we tried is fuzzy string matching of all the brands available in our dictionary. But this will lead to a lot of false-positive results which lead to a decrease in our f1 score.

Sentiment Analysis

Once we identify the brands, the ***aspect-based sentiment classification model*** aims at identifying the sentiment polarities of aspects (brands) explicitly given in sentences.

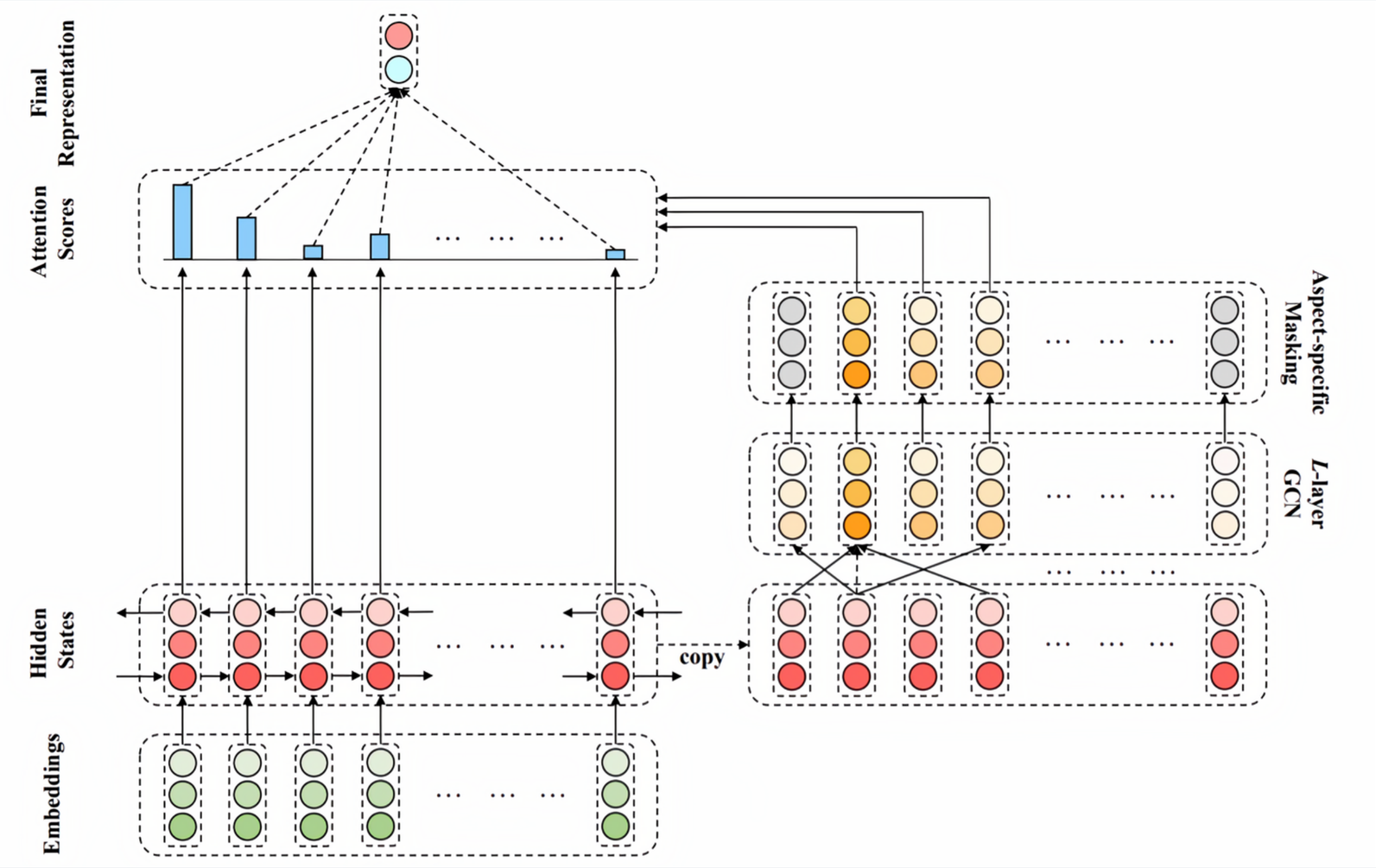
The Dataset

- LAP14 (Pontiki et al., 2014): Dataset with laptop-related reviews.
- Scraped Articles from NDTV Gadgets (for mobile tech) and random news articles (non-mobile tech).
- 600 + 3000 Articles
- Scraped Tweets
- 3000 Tweets

Our approach

To account for ***relevant syntactic constraints and long-range word dependencies***, we use a Graph Convolutional Network (GCN) over a sentence's dependency tree to exploit syntactic information and word dependencies.

Machine Learning Model



Embedding and Bidirectional LSTM:
With the word embeddings of the sentence obtained from Glove, a bi-LSTM is constructed to produce hidden state vectors.

Obtaining Aspect-oriented Features:
we got aspect-oriented features by applying multi-layer graph convolution over the syntactical dependency tree of a sentence and imposing an aspect-specific masking layer on its top.

Aspect-specific Masking:
We mask out hidden state vectors of non-aspect words and keep the aspect word states unchanged. Through graph convolution, these features mask can capture long term dependencies

Aspect-aware Attention:
Using the aspect-oriented features, a refined representation of the hidden state vectors is produced. The idea is to retrieve significant features that are semantically relevant to the aspect words from the hidden state vectors and set a retrieval-based attention weight for each context word.

Results

Train F1	0.90
Primary Validation F1	0.65
Secondary Validation F1	0.77
Test F1	0.48

Headline Generation

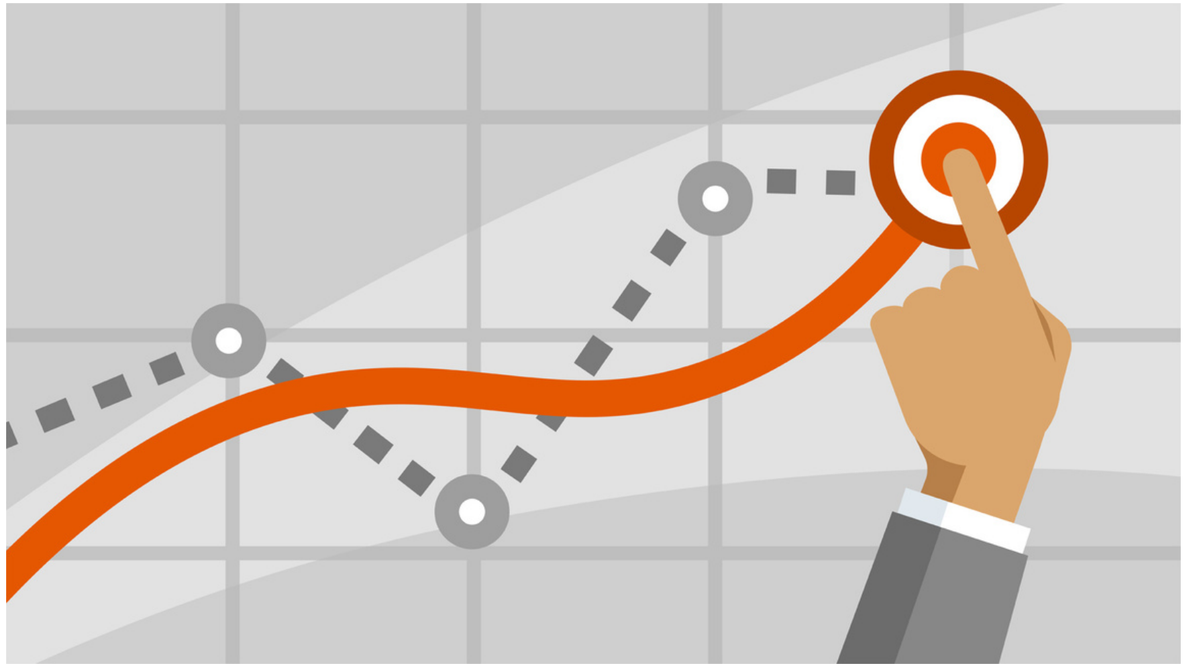
The final task of the pipeline is to generate the heading of the articles classified as mobile tech.

Our approach

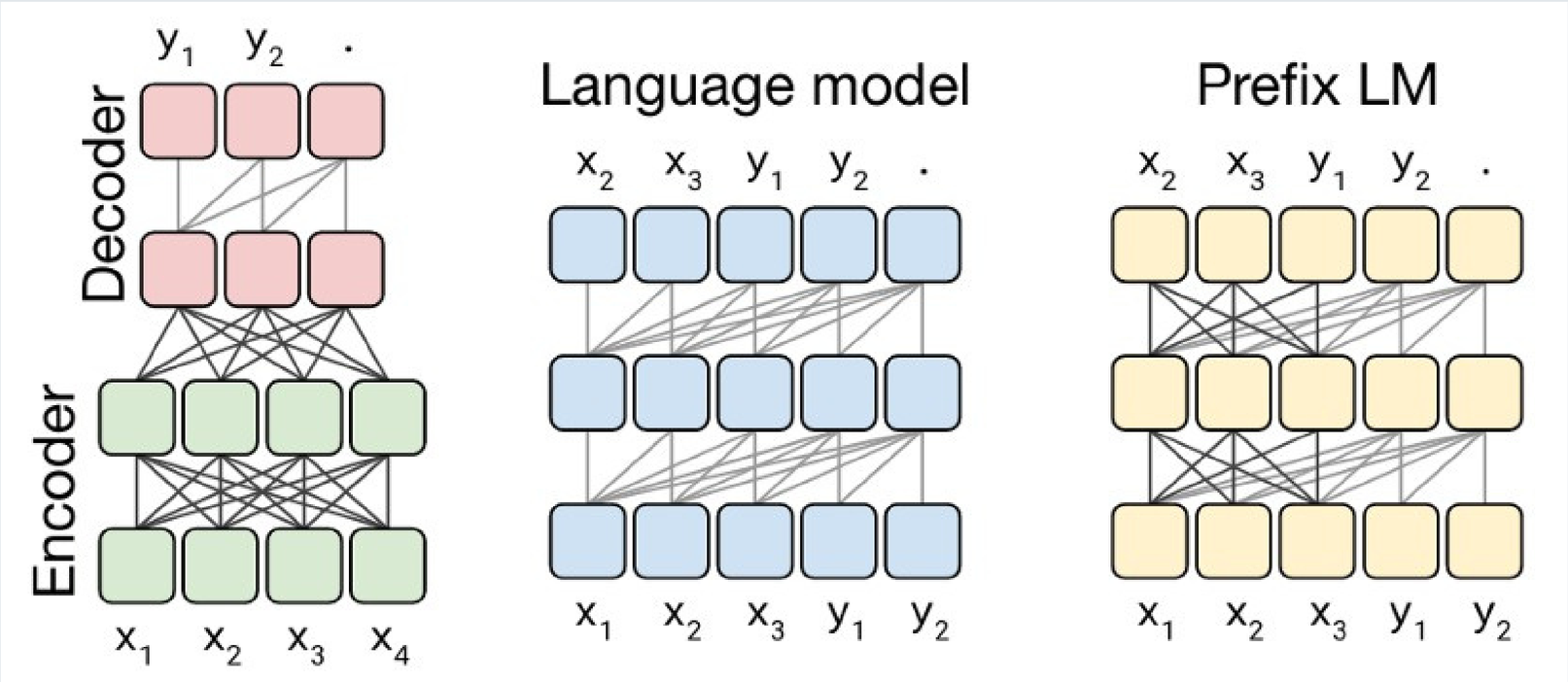
1. Extractive Summary: The network calculates the most important sentences from the article and gets them together to provide the most meaningful information from the article.

2. Abstractive Summary: The network creates new sentences to encapsulate the maximum gist of the article and generates that as output. The sentences in the summary may or may not be contained in the article.

We generated an Abstractive Summary.



Machine Learning Model



T5 as a Language Model

- We have used one of the most recent and novel transformers model T5.
- T5 in many ways is one of its kind transformers architecture that not only gives state-of-the-art results in many NLP tasks but also has a very radical approach to NLP tasks.
- Text-2-Text: According to the graphic taken from the T5 paper. All NLP tasks are converted to a text-to-text problem. Tasks such as translation, classification, summarization, and question answering, all are treated as text-to-text conversion problems, rather than seen as separate unique problem statements.

Why T5 Small?

- The T5 model parameters were less than the BERT transformer for text summarisation and **performed better** than the BERT in our dataset.
- T5 is an **encoder-decoder model** and converts all NLP problems into a **text-to-text format**.
- For the text summarization, T5 was trained on the CNN/Daily Mail dataset which is posed as an abstractive summarization benchmark.
- The BERT base had ***110 million parameters*** but the T5 small we used had just ***65 million parameters***.

Results

Train score	0.83
Primary Validation score	0.47
Secondary Validation score	0.65
Test score	0.27

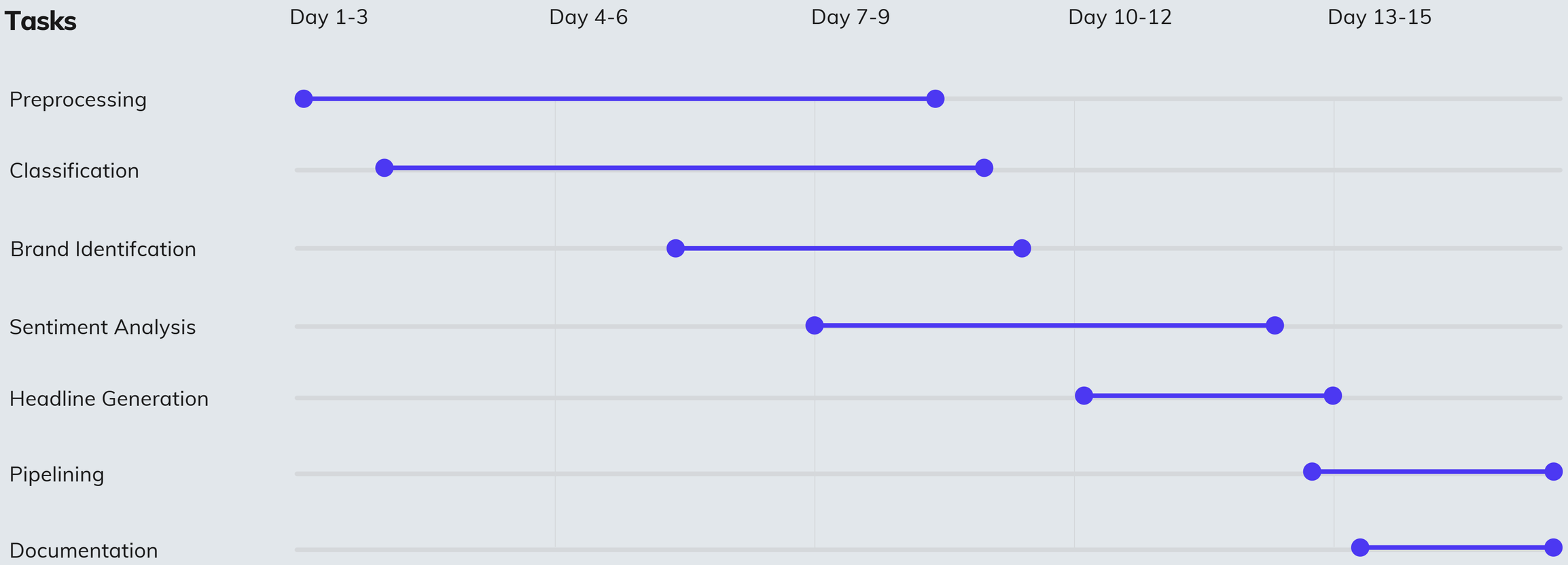
Scope For Improvement

Using Larger Beam Search

We could have used much larger beam search although it takes more time it will give better results.

Trying other better models

We have used the T5-small model if we would have used the T5-base it could have performed much better but for training, we need better GPU.


Tasks
Day 1-3
Day 4-6
Day 7-9
Day 10-12
Day 13-15
Preprocessing
Classification
Brand Identifcation
Sentiment Analysis
Headline Generation
Pipelining
Documentation

Thank you!