Carnegie Mellon University

Setting Expectations

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

Setting expectations – final exam

- The semester ends on December 6. Final exam week is December 9-13.
- We’ll know the final exam date once SCS announces it.
- CMU students are expected to be on campus through the final exam week.
- Barring medical issues, there will be no accommodations for missing the final exam.

The semester ends on December 6. Final exam week is December 9-13.
We’ll know the final exam date once SCS announces it.
CMU students are expected to be on campus through the final exam week.
Barring medical issues, there will be no accommodations for missing the final exam.

Setting expectations – Piazza

- TAs are expected to work weekdays only.
- We aim for a Piazza response time of 1-2 business days.
- If you don't get a response within 2 business days, you may reach out by email to escalate.

TAs are expected to work weekdays only.
We aim for a Piazza response time of 1-2 business days.
If you don't get a response within 2 business days, you may reach out by email to escalate.

Setting expectations – coming to class

- We are going to be releasing class recordings ~1 week after class happens.
- Get the most out of your experience by coming in-person to class.
- When you all participate in class:
  - we can adjust our lectures on the fly to better teach you.
  - we learn what to emphasize in future lectures.
- Your in-class participation makes the class better!

QUIZ

Learning Objectives

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

What I’ve told you so far:

Language models are trained with the objective of predicting the next word in a sequence given the previous words (and possibly some other conditioning signal).

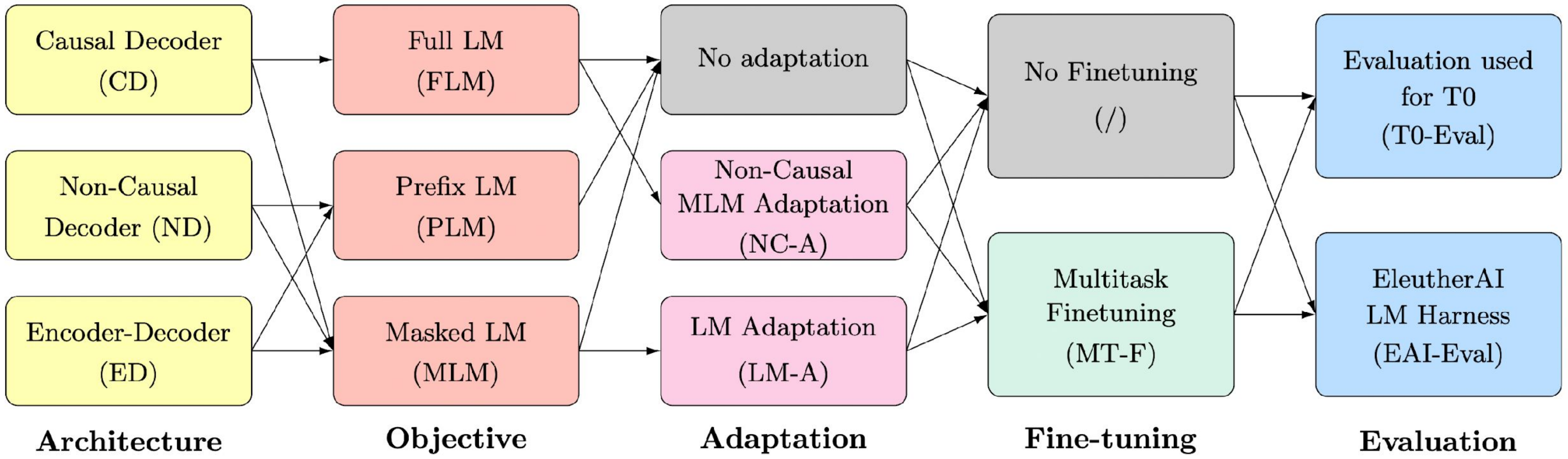
Possible objectives for pre-training an encoder-decoder model:

- Predict a suffix given a prefix.
  - Input: ***I took my dog, Fido, to the***
  - Target**:** ***park for his walk.***
- Masked language modeling
  - Input: ***I took <x> to <y> his walk.***
  - Target: ***<x> my dog, Fido, <y> the park for***

Possible objectives for pre-training a decoder-only model:

- Predict a suffix given a prefix.
    - Input: ***I took my dog, Fido, to the***
    - Target: ***park for his walk.***
- Masked language modeling
    - Input: ***I took <x> to <y> his walk.***
    - Target: ***<x> my dog, Fido, <y> the park for***
- Full language modeling: predict next token given previous tokens
    - Target: ***I took my dog, Fido, to the park for his walk.***

It is also possible to mix and match different training objectives.



Wang et al. "What Language Model Architecture and Pretraining Objective Work Best for Zero-Shot Generalization?"

Turning Text into Pretraining Data

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

Preprocessing

Data **preprocessing** is the pipeline to turn raw data (in whatever format it has been give to you) into data that is ready to be ingested by the neural network.

Ideal scenario: start off with fairly clean text

- When this might happen
  - Wikipedia, Books, datasets meant for classic NLP applications
  - High-resource languages: English, French, etc.
- A perfect world situation: Texts are clean and well-formatted
  - (Unlikely to achieve in real-world systems)

Preprocessing Clean Texts

Since the text is already cleaned, all we need to do is convert it into batches of training data.

The Steelers enjoy a large, widespread fanbase nicknamed Steeler Nation. They currently play their home games at Acrisure Stadium.

Raw Clean Text

Tokenization →

'_The', '_Steel', 'ers', '_enjoy', '_a', '_large', ',', '_wide', 'spre', 'ad', '_fan', 'base', '_nick', 'na', 'med', '_Steel', 'er', '_Nation', '.', '_They', '_currently', '_play', '_their', '_home', '_games', '_at', '_A', 'cris', 'ure', '_Stadium', '.'

Tokenized

Batching →

[580, 109027, 1313, 25224, 9, 21333, 3, 38133, 21328, 711, 1206, 37381, 128910, 75, 4805, 109027, 55, 82580, 4, 0]
[10659, 82423, 11300, 2362, 5367, 27527, 98, 61, 58531, 3407, 88259, 4,0,0,0,0,0,0,0,0]

Tensor

Tokenizing Text

A **tokenizer** takes text and turns it into a sequence of discrete **tokens.**

A **vocabulary** is the list of all available tokens.

Let's tokenize: "A hippopotamus ate my homework."

Vocab Type	Example	Ex. length
character-level	['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's', ' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']	31
subword-level	['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']	9
word-level	['A', 'hippopotamus', 'ate', 'my', 'homework', '.']	5

Word-level Tokenization

Method: rule-based---split text by spaces, punctuation, and other hand-written rules.

Challenges:

- Open vocabulary problem
    - Many words may never appear in training data. They become [UNK].
    - This is more several in some languages, e.g. languages that concatenate words.
- Typo'ed words also get tokenized to _UNK

Subword Tokenization

Method: words get split into multiple tokens.

Advantages:

- Vocabulary is built dynamically
    - Frequent words get assigned their own token
    - Rare words are split into sub-words.

Subword Tokenization: Byte Pair Encoding (BPE)

Main Idea:

- Construct subword vocabulary by learning to merge characters
- Inspiration comes from compression algorithms

Training Steps:

1. Initialize your vocabulary with every character as a token.
    - E.g., in English, alphabet + numbers + punctuation
2. Merge the most frequent token pair in your corpus.
    - Vocabulary size +1
3. Re-tokenize the corpus with the merged subword pair
    - That is, merge all appearances of the pair, and replace with the new token.
4. Repeat step 2-3 until the target vocabulary size is reached.

Subword Tokenization: Byte Pair Encoding (BPE)

An efficient learning implementation of BPE

1. Take your corpus and split it into words (e.g. using a rule-based tokenizer)
2. Construct a word dictionary with frequency counts

   ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

3. Start from uni-character vocabulary, merge pairs by frequency, till reached target vocabulary size

Subword Tokenization: Byte Pair Encoding (BPE)

An efficient learning implementation of BPE

1. Take your corpus and split it into words (e.g. using a rule-based tokenizer)
2. Construct a word dictionary with frequency counts

   ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

3. Start from uni-character vocabulary, merge pairs by frequency, till reached target vocabulary size

Iteration	Current Vocab	Tokenization of Words
1	*["b", "g", "h", "n", "p", "s", "u"]*	*("h" <u>"u" "g"</u>, 10), ("p" <u>"u" "g"</u>, 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" <u>"u" "g"</u> "s", 5)*

Subword Tokenization: Byte Pair Encoding (BPE)

An efficient learning implementation of BPE

1. Take your corpus and split it into words (e.g. using a rule-based tokenizer)
2. Construct a word dictionary with frequency counts

   ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

3. Start from uni-character vocabulary, merge pairs by frequency, till reached target vocabulary size

Iteration	Current Vocab	Tokenization of Words
1	["b", "g", "h", "n", "p", "s", "u"]	("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
2	["b", "g", "h", "n", "p", "s", "u", "ug"]	("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)

Subword Tokenization: Byte Pair Encoding (BPE)

An efficient learning implementation of BPE

1. Take your corpus and split it into words (e.g. using a rule-based tokenizer)
2. Construct a word dictionary with frequency counts

   ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

3. Start from uni-character vocabulary, merge pairs by frequency, till reached target vocabulary size

Iteration	Current Vocab	Tokenization of Words
1	["b", "g", "h", "n", "p", "s", "u"]	("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
2	["b", "g", "h", "n", "p", "s", "u", "ug"]	("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
3	["b", "g", "h", "n", "p", "s", "u", "ug" "un"]	("h" "ug", 10), ("p" "ug", 5), ("p" "un" , 12), ("b" "un", 4), ("h" "ug" "s", 5)

Subword Tokenization: Byte Pair Encoding (BPE)

An efficient learning implementation of BPE

1. Take your corpus and split it into words (e.g. using a rule-based tokenizer)
2. Construct a word dictionary with frequency counts

   ("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

3. Start from uni-character vocabulary, merge pairs by frequency, till reached target vocabulary size

Iteration	Current Vocab	Tokenization of Words
1	["b", "g", "h", "n", "p", "s", "u"]	("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
2	["b", "g", "h", "n", "p", "s", "u", "ug"]	("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
3	["b", "g", "h", "n", "p", "s", "u", "ug" "un"]	("h" "ug", 10), ("p" "ug", 5), ("p" "un" , 12), ("b" "un", 4), ("h" "ug" "s", 5)
4	["b", "g", "h", "n", "p", "s", "u", "ug" "un", "hug"]	("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)

Subword Tokenization: Pros

Controlled vocabulary size

- A pre-defined hyperparameter as a design choice

Learned vocabulary, best compromise between character-level and word-level tokenization.

- Frequent words kept whole
- Tail words split to sub-words
    - More observations on sub-words
    - Utilization of morphology information

Subword Tokenization: Pros

Controlled vocabulary size

- A pre-defined hyperparameter as a design choice

Learned vocabulary, best compromise between character-level and word-level tokenization.

- Frequent words kept whole
- Tail words split to sub-words
  - More observations on sub-words
  - Utilization of morphology information

Units are well-suited to neural methods

- Trivial for Transformers to figure out common combinations
- Neural representations smooth rare combinations

Subword Tokenization: Further Quandries

When treating characters as your basic units, unknown (sub)tokens can still exist.

- Example: If your basic units are [A-Za-z], Chinese characters can't be tokenized.
- Solution: Byte-level BPE that uses raw bytes (e.g., Unicode bytes) as the basic character set.

Subword Tokenization: Further Quandries

When treating characters as your basic units, unknown (sub)tokens can still exist.

- Example: If your basic units are [A-Za-z], Chinese characters can't be tokenized.
- Solution: Byte-level BPE that uses raw bytes (e.g., Unicode bytes) as the basic character set.

Algorithm requires an initial word-level tokenization. This can be tricky.

- Example: In Chinese and in code, splitting the text by spaces won't work well.
- Solution: More clever rule-based systems for performing initial tokenization.

Subword Tokenization: Further Quandries

When treating characters as your basic units, unknown (sub)tokens can still exist.
- Example: If your basic units are [A-Za-z], Chinese characters can't be tokenized.
- Solution: Byte-level BPE that uses raw bytes (e.g., Unicode bytes) as the basic character set.

Algorithm requires an initial word-level tokenization. This can be tricky.
- Example: In Chinese and in code, splitting the text by spaces won't work well.
- Solution: More clever rule-based systems for performing initial tokenization.

Step of finding most frequent pairs has a naive runtime of $O(n^2)$
- Why: For each possible combination of two tokens, count number of occurrences.
- Solution: Various algorithmic and implementation improvements, such as caching occurrence counts.

Preprocessing Clean Texts

Since the text is already cleaned, all we need to do is convert it into batches of training data.

The Steelers enjoy a large, widespread fanbase nicknamed Steeler Nation. They currently play their home games at Acrisure Stadium.

Raw Clean Text

Tokenization →

'_The', '_Steel', 'ers', '_enjoy', '_a', '_large', ',', '_wide', 'spre', 'ad', '_fan', 'base', '_nick', 'na', 'med', '_Steel', 'er', '_Nation', '.', '_They', '_currently', '_play', '_their', '_home', '_games', '_at', '_A', 'cris', 'ure', '_Stadium', '.'

Tokenized

Batching →

[580, 109027, 1313, 25224, 9, 21333, 3, 38133, 21328, 711, 1206, 37381, 128910, 75, 4805, 109027, 55, 32580, 4, 0]
[10659, 82423, 11300, 2362, 5367, 27527, 98, 61, 58531, 3407, 88259, 4,0,0,0,0,0,0,0,0]

Tensor

What is batch size?

- Number of examples
- Number of sentences/paragraphs
- Number of documents
- Number of tokens

What is batch size?

- Number of examples
- Number of sentences/paragraphs
- Number of documents
- Number of tokens

Batching Data

How can we batch the following?

"Pittsburgh, Pennsylvania"	➡	[47, 1468, 26320, 11, 20355]
"Fido the dog likes eating rabbit-flavored ice cream	➡	[37, 5362, 279, 5679, 13452, 12459, 39824, 12556, 76486, 10054, 12932, 13]
"It is raining outside"	➡	[2181, 374, 84353, 4994]

Let's assume a batch-size of 8.

Batching Data

How can we batch the following?

"Pittsburgh, Pennsylvania"	➡	[47, 1468, 26320, 11, 20355]
"Fido the dog likes eating rabbit-flavored ice cream	➡	[37, 5362, 279, 5679, 13452, 12459, 39824, 12556, 76486, 10054, 12932, 13]
"It is raining outside"	➡	[2181, 374, 84353, 4994]

Let's assume a batch-size of 8.

OPTION 1: Each document gets placed into its own batch.

Batch_0 = [47, 1468, 26320, 11, 20355, 0, 0, 0]
Batch_1 = [37, 5362, 279, 5679, 13452, 12459, 39824, 12556]
Batch_3 = [2181, 374 , 84353, 4994, 0, 0, 0, 0]

Batching Data

How can we batch the following?

"Pittsburgh, Pennsylvania"	➡	[47, 1468, 26320, 11, 20355]
"Fido the dog likes eating rabbit-flavored ice cream	➡	[37, 5362, 279, 5679, 13452, 12459, 39824, 12556, 76486, 10054, 12932, 13]
"It is raining outside"	➡	[2181, 374, 84353, 4994]

Let's assume a batch-size of 8.

OPTION 2: Documents get patched into batches.

Batch_0 = [47, 1468, 26320, 11, 20355, 0, 37, 5362]
Batch_1 = [279, 5679, 13452, 12459, 39824, 12556, 76486, 10054]
Batch_3 = [12932, 13, 0, 2181, 374 , 84353, 4994, 0, 55]

What is batch size?

- Number of examples
  - Makes sense in computer vision when examples are all the same shape.
  - Doesn't make sure for language models.
- Number of sentences/paragraphs
  - This would require parsing the text.
  - Same issues with variable length.
- Number of documents
  - We could do one document per batch, but padding is inefficient.
- Number of tokens
  - This is what people tend to mean when they refer to batches during pre-training.
  - Common batch sizes:
    - LLaMA-1: 4 million tokens
    - T5: 65,536 tokens
    - Olmo: 4.19 M tokens
    - GPT-3 175B: 3.2M tokens

What is batch size?

- Number of examples
  - Makes sense in computer vision when examples are all the same shape.
  - Doesn't make sure for language models.
- Number of sentences/paragraphs
  - This would require parsing the text.
  - Same issues with variable length.
- Number of documents
  - We could do one document per batch, but padding is inefficient.
- Number of tokens
  - This is what people tend to mean when they refer to batches during pre-training.
  - Common batch sizes:
    - LLaMA-1: 4 million tokens
    - T5: 65,536 tokens
    - Olmo: 4.19 M tokens
    - GPT-3 175B: 3.2M tokens

There are lots of complexities around batch size and multi-GPU parallelism I'm glossing over.

Scaling up Pretraining Data

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

What are the goals of pre-training?

The goals of **pre-training** are to get the language model to:

- learn the structure of natural language
- learn humans' understanding of the world (as encoded in the training data).

- Goal:

What are the goals of pre-training?

The goals of **pre-training** are to get the language model to:

- learn the structure of natural language
- learn humans’ understanding of the world (as encoded in the training data).

Ideal data for pre-training:

- Data that is high-quality, clean, and diverse.
- Books, Wikipedia, news, scientific papers, etc.

Pre-training data reality

More data is almost always better.

Table 11: $\text{RoBERTa}_{\text{Large}}$ Pretrained with Different Data Setups[10].

Pretrain Setups	Data Size	Batch Size	Steps	MNLI-m (ACC)	SQuAD 1.1 (F1)
Wiki + Books	16GB	8K	100K	89.0	93.6
+Additional Data	160GB	8K	100K	89.3	94.0
+Pretrain More Steps	160GB	8K	300K	90.0	94.4
+Even More Steps	160GB	8K	500K	90.2	94.6

- Same model, same pretraining steps, more data helps
- With more data more pretraining steps also help
- Empirical gains more than many "fancier" improvements

Li, et al. "RoBERTa: A robustly optimized BeRT pretraining approach." arXiv preprint arXiv:1907.11692. 2019.

Pre-training Data Reality

High quality data eventually runs out.

In practice, the web is the most viable option for data collection.

- In the digital era, this is the go-to place for general domain human knowledge.
- It is massive and unlikely to grow slower than computing resources*
- Publicly available*

*. More on how true these points are later in the class.

Pre-training Data Reality

Web data is plentiful, but can be challenging to work with.

- Copyright and usage constraints can get extremely complicated
- Data is noisy, dirty, and biased
- Data is contaminated with auto-generated text
    - Not just from LLM usage, but also tons of templated text.

The Web Data Pipeline

1. Content is posted to the web.

2. Webcrawlers identify and download a portion of this content.

3. The data is filtered and cleaned.

The Web Data Pipeline

1. Content is posted to the web.

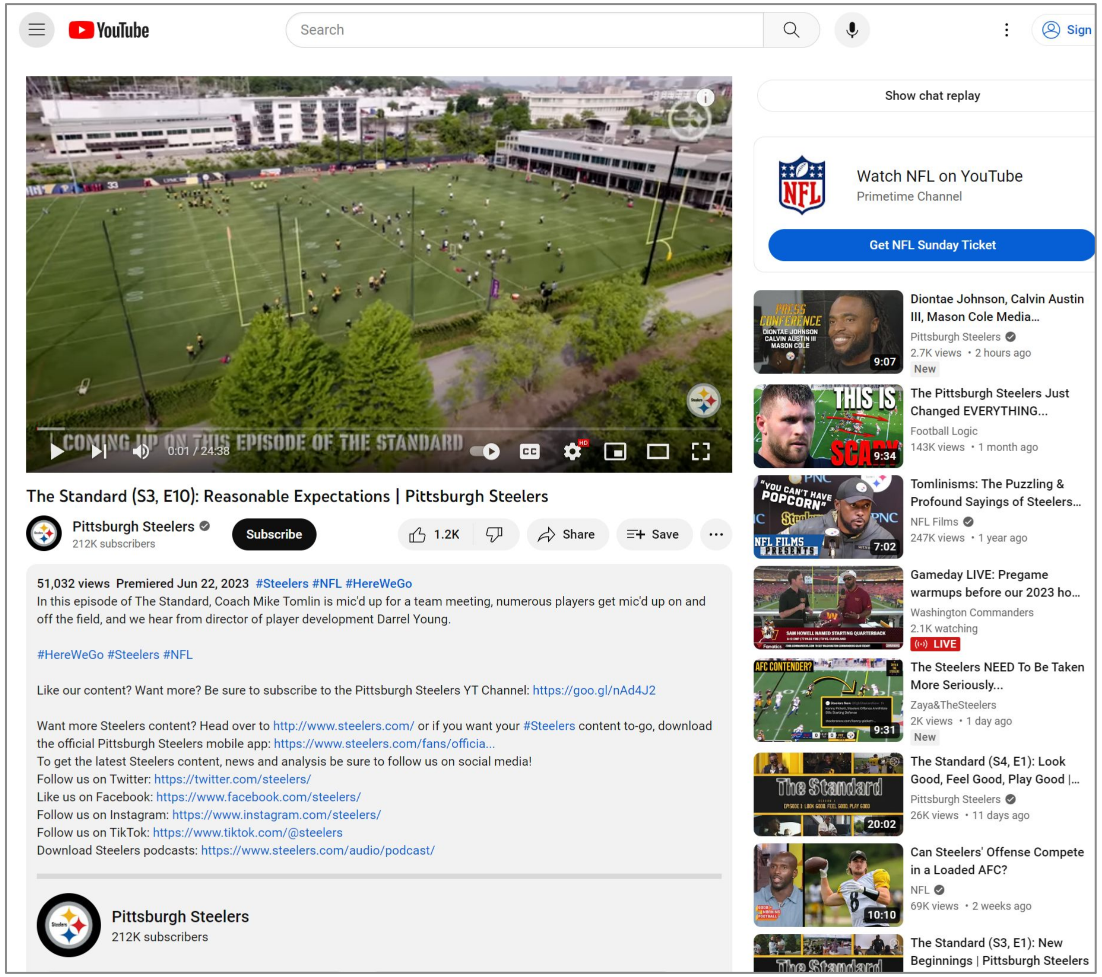
2. Webcrawlers identify and download a portion of this content.

3. The data is filtered and cleaned.

1. Content is posted to the internet.

Challenges

- Biases in what's available:
  - Recency bias
  - Demographic biases
  - Language biases
- Web is much more dynamic than static HTML pages
  - CSS, JavaScript, interactivity, etc.
  - Responsive design
  - Many HTML pages involves 20+ secondary URLs, iframes, etc.
- What counts as content?
  - Ads, recommendation, navigation, etc.
  - Multimedia: images, videos, tables, etc.
  - Spam



Goal:

1. Content is posted to the internet.

Content extraction from webpages is a well-studied problem in industry.

- Can be very engineering and resource heavy to do well.
- Existing toolkits is a strategic advantage of some proprietary LLMs

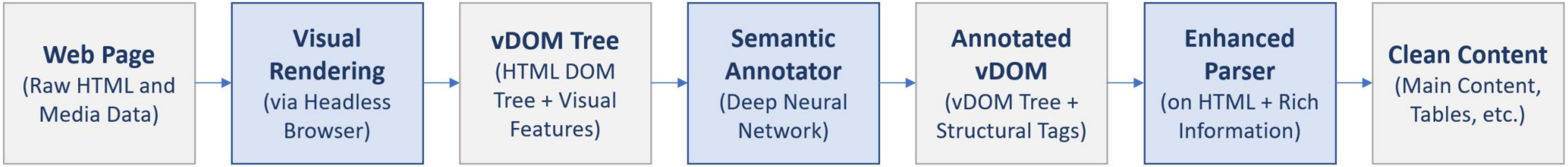


Figure 8: A Content Extraction Pipeline from Bing Used for ClueWeb22

The Web Data Pipeline

1. Content is posted to the web.

2. Webcrawlers identify and download a portion of this content.

3. The data is filtered and cleaned.

2. Webcrawlers identify and download a portion of this content.

General Idea

1. Start with a set of seed websites
2. Explore outward by following all hyperlinks on the webpage.
3. Systematically download each webpage and extract the raw text.

2. Webcrawlers identify and download a portion of this content.

General Idea

1. Start with a set of seed websites
2. Explore outward by following all hyperlinks on the webpage.
3. Systematically download each webpage and extract the raw text.

Challenges

- How to harvest a large number of seed URLs efficiently
- How to select “high quality” URLs and skip over “bad” URLs
  - Some cases are clear cut: spammy, unsafe, etc.
  - Some are hard to detect or up to debate: certain biases.
- How to keep the crawl up-to-date
  - Given a fixed compute budget each month, is it better to crawl new webpages, or recrawl old ones that might’ve changed?

The Web Data Pipeline

1. Content is posted to the web.

2. Webcrawlers identify and download a portion of this content.

3. The data is filtered and cleaned.

3. The data is filtered and cleaned.

Remove noisy, spammy, templated, and and fragmented texts

- These portions lack the content needed to meet pretraining goals.

Select higher quality texts from a massive candidate pool

- Given a limited pretraining compute budget, we'd like pretrain on better texts

Avoid toxic and biased content

- NSFW content
- Texts with strong biases

3. The data is filtered and cleaned.

Methods to identify high-quality content:

- Rule-based heuristics for quality
  - You'll be implementing a bunch of these in the homework
- Proximity to known high-quality indicators
  - E.g. a website that was highly upvoted on Reddit, or a website that was included as a reference on Wikipedia
- Classifiers
  - Trained quality and toxicity classifiers are common.
  - E.g. train a classifier with Wikipeida as the positive examples and random web pages as the negative examples

3. The data is filtered and cleaned.

Example: Rule-based filtering in C4 (pre-training dataset for T5 model)

1. Start from Common Crawl's official extracted texts from HTML
2. Only keep text lines ended with a terminal punctuation mark
3. Discard pages with fewer than 5 sentences
4. Only keep lines with at least 3 words
5. Remove any line with the word "Javascript"
6. Remove any page
    1. with any words in a toxic word dictionary
    2. with the phrase "lorem ipsum"
    3. With "{"
7. De-dup at three-sentence span level

3. The data is filtered and cleaned.

Challenges

- At what granularity should filtering be performed?
  - Word-level, sentence-level, paragraph-level, document level?
- What constitutes “high quality” or “non-toxic”?
- Are our filters/classifiers multilingual? Even within English, do they treat all groups equally?
- It is very expensive to ablate pre-training dataset decisions.

A Couple Dataset Case Studies

Large Language Models: Methods and Applications

Daphne Ippolito and Chenyan Xiong

Case Study: WebText

WebText: The pretraining corpus of GPT-2

- Harvested all outbound links from Reddit
  - These are all URLs that were manually mentioned by humans
- Only kept links that received >= 3 “Karma”
- Deduplicated URLs
  - Total 45 million URLs deduped to 8 million web pages

Case Study: WebText

Web exploration method

How to harvest URLs:

- From Reddit mentions

Definition of good URLs:

- Other Reddit users like it

- Pros
  - Easy to harvest a relatively large set of URLs from a common resource
  - Human votes on the URLs
- Cons
  - Since then, Reddit has forbid the use of its data for pretraining LLMs
  - Limited scale
  - Reddit is not the cleanest part of the internet

Case Study: CommonCrawl

Common Crawl: commoncrawl.org

- Non-profit organization provides **open** access to large scale web crawls
- Petabytes of web pages available
- Monthly crawls and dumps
  - Re-crawled web pages and fresh dumps (bi)monthly
  - Recent dumps are ~3 billion pages
  - Date back to past 10 years
    - "220 billion web pages (HTML) captured 2008 – 2021"

Case Study: CommonCrawl

Web exploration approach

- Start from a set of seed URLs: popular, high-quality, and trustworthy websites
  - Gov, Edu, etc.
  - Top web domains
- Traverse the web to obtain a candidate set of URLs
  - Around 500 billion links discovered per month crawl
  - About 25+ billion unique ones
- Prioritizes a subset (~3 billion) of URLs to crawl and include in the dump
  - Does this to make the best use of crawling budget.

Case Study: CommonCrawl

Which URLs to crawl?

- 2008-2012: CC's in-house Page Rank
- 2012-2015: Added ranking and metadata of 22 billion pages donated from web search engine blekko
- 2016-2018: Occasional seed URL donations, ~400 million URLs
- 2016: Alexa and Common Search Rankings
- 2017-Now: CC's in-house web graph based rankings (page rank and centrality)
    - Importance score calculated based on past three-month dumps
    - Steer the crawler for next three month
    - Capped # of URLs per domain

Case Study: CommonCrawl

Pros

- The one and the only public web crawl at this scale
  - Still far away from commercial search engines, but the closest we have
- 10 years of crawl dumps have enabled various data subsamples
- Accumulation of low resource languages
  - One can combine low resources languages from years of dumps to pair with English texts from one month

Cons

Each crawl is ~3billion documents, still limited coverage of the massive web

- Crawls every month restart from the seed URLs
  - Crawls never grow above ~3B documents
- URL distributions skewed by crawling prioritization, per domain URL cap, and physical location of the crawling machines (and people)

Questions?