Large Language Models for Text Classification

Korea University Methods Workshop

Taegyoon Kim | DHCSS, KAIST

Jan 22, 2026

Agenda

Things to be covered

- What is text classification?
- Evolution of text classification
- The classification pipeline
- Fine-tuning representation models
  - How BERT is adapted for classification
- Generative LLMs for classification
  - Prompting, decoding, and trade-offs
- Guided coding

Supervised Learning for Text Classification

We will focus on text classification with supervised learning

- Goal
  - To classify documents into pre-defined categories
  - E.g., sentiment of social media comments (e.g., positive-negative), politicians’ stances on policy issues (e.g., for-against-neutral on immigration), etc.
- We need
  - Labeled data set (for training and/or testing)
  - Model that maps texts to labels
  - Evaluation approaches: performance metrics, cross-validation, etc.

Supervised Learning for Text Classification

Two fundamental approaches in machine learning

	Supervised	**Unsupervised**
Common tasks	Regression, classification	Clustering, dimensionality reduction
Objective	Trained on a labeled data to learn a mapping from input to output	Find patterns or structures within data without labeled data
Outcome	Pre-defined categories	Not quite pre-defined
Model evaluation	Explicit metrics such as accuracy, precision, recall, or MSE	Can involve qualitative assessment

Supervised Learning for Text Classification

Evolution of text classification

- Dictionary methods
  - Based on counting (or weighting) of relevant keywords
  - Readily available and fast ↔ sub-optimal performance
  - However, if you want to quick, preliminary analysis for concepts supported by an existing dictionary, it can be helpful
  - E.g., LIWC, VADER, Moral Foundations Dictionaries, etc.

Supervised Learning for Text Classification

Evolution of text classification (cont'd)

- Traditional ML algorithms
  - This approach provides the groundwork for many foundational concepts in text classification
  - Classifiers (models) are trained to learn the relationships between texts and labels (i.e., classes)
  - So this requires training (labeled) data (pairs of a text and a label)
  - (On average) more training data, higher performance
  - E.g., logistic regression, random forest, SVM, neural networks

Supervised Learning

Evolution of text classification (cont’d)

- Fine-tuning representation models (e.g., BERT family)
    - These models are pre-trained with massive amounts of text data
    - Given a text, representation models encode it into a vector (an array of numbers) that captures its meaning (thus *represent*)
    - We can *fine-tune* such a model for classification
    - They tend to achieve higher performance than the traditional ML approaches
    - It might still require (potentially large) labeled training data for high performance

Supervised Learning for Text Classification

Evolution of text classification (cont’d)

- Prompting generative models (e.g., GPT family)
  - Like representation models, these models are pre-trained on vast amounts of text data, often even more extensively
  - Generative models are designed to generate text outputs given input text prompts (unlike representation models)
  - We can prompt such a model with unlabeled texts to generate labels
  - Still requires labeled data: not for training but for testing performance

Overview of Process

Three-step procedure

- Step 1: building a labeled data set
- Step 2: training model(s)
- Step 3: evaluating performance

Overview of Process

Three-step procedure

- **Step 1: building a labeled data set**
- Step 2: training model(s)
- **Step 3: evaluating performance**

$\ast$ Steps 1 and 3 apply to all of the classification approaches

Overview of Process

Step 1: build a labeled data set

- Label texts following systematic labeling guidelines and check inter-coder reliability (e.g., Krippendorff’s α)
- Iterative process
  - Oftentimes, definition/operationalization does not often take place at once but in an iterative process
  - In many cases, it is difficult to specify an entire annotation guidelines *ex ante*
  - Preliminary labeling rule are written and applied to an initial set of docs
    $\rightarrow$ Annotators identify ambiguities in the rule
    $\rightarrow$ Revision of the rule $\rightarrow \dots$
- The labeled data set (C) will serve as “ground-truth” or “gold standards”
- C is used to training (building a model) and/or validation (evaluating the model’s performance)

Overview of Process

Step 1: build a labeled data set

Doc number	Text	y
1	This is great!	0
2	$\%^{@\%}$ off!	1
…		
9999	This is sick	0
10000	Love BTS <3	0

Overview of Process

Step 1: build a labeled data set

Doc number	Text	y
1	This is great!	0
2	%@% off!	1
...		
9999	This is sick	0
10000	Love BTS <3	0

Overview of Process

Step 2: train models

- Applies to training traditional ML models and fine-turning representation models
- Does *not* apply to prompting generative models
- Randomly split C into a training set (C_{train}) and a test set (C_{test})
  - Typically, $C_{train} : C_{test}$ = 7:3 or 8:2
  - E.g., identifying YouTube comments containing hate speech
    - C: 10,000 comments labeled for the presence of hate speech
    - $C_{training}$: 8,000 comments for training
    - C_{test}: 2,000 comments for test

Overview of Process

Step 2: train models

Index	y	Feature 1	Feature 2	…	Feature V-1	Feature V
1	0	3	1.4	…	1.7	6
2	1	-0.8	6.4	…	5.7	-1.6
…						
7999	0	-2.8	0.9	…	3.3	-0.6
8000	0	3.7	1.4	…	5.7	-5.8

Overview of Process

Step 2: train models

- Choose a model F (e.g., logistic regression / BERT) and learn/fine-tune model parameters β
  - The model provides a mapping between X_{train} and y_{train}
- Loss (cost) function: measures how much model predictions ($\hat{y}_{train}$) differ from the true labels (y_{train})
  - $\hat{y}_{train} = F(\hat{\beta} * X_{train})$
  - β is estimated in a way that minimizes the difference between the two
- As a result, we get a classifier, $F(\hat{\beta} * X)$, which generates predictions, $\hat{y}$

Overview of Process

Step 2: train models

Index	**y**	**$\hat{y}$**	**Feature 1**	**Feature 2**	**…**	**Feature V-1**	**Feature V**
1	0	1	3	1.4	…	1.7	6
2	1	1	-0.8	6.4	…	5.7	-1.6
…							
7999	0	0	-2.8	0.9	…	3.3	-0.6
8000	0	0	3.7	1.4	…	5.7	-5.8

Overview of Process

Step 3: evaluate performance

- We held out another labeled set C_{test} (n = 2,000) (to avoid overfitting)
- Use the classifier $F(\hat{\beta} * X)$ to generate predictions $\hat{y}_{test}$
- Compare the predictions $\hat{y}_{test}$ and the true labels y_{test}
- Performance metrics include accuracy, precision, recall, etc.
- (Then use the classifier for unlabeled data)

Overview of Process

Step 3: evaluate performance

Index	**y**	**$\hat{y}$**	**Feature 1**	**Feature 2**	**…**	**Feature V-1**	**Feature V**
1	1	1	3.12	1.99	…	5.77	0.36
2	1	0	-0.8	1.14	…	9.71	-1.66
…							
1999	0	0	-2.11	0.95	…	1.23	-0.62
2000	0	0	3.71	1.48	…	1.7	-5.84

Overview of Process

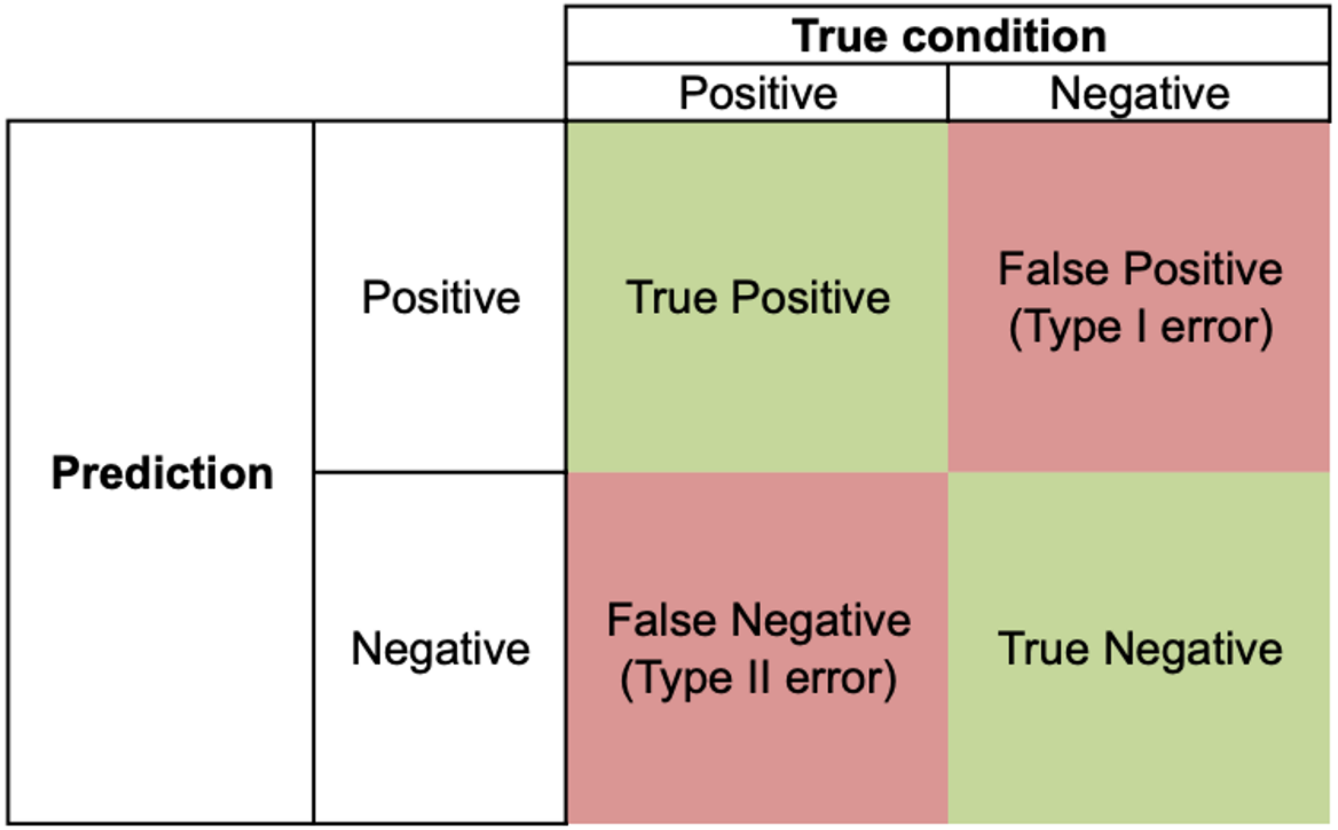
Step 3: evaluate performance

- Performance metrics
  - Accuracy: the proportion of all predictions (both positive and negative) that the model got right
  - Precision: the proportion of positive predictions that were actually correct
  - Recall: the proportion of actual positives that were correctly predicted
  - F-1: the harmonic (as opposed to arithmetic) mean of precision and recall

Overview of Process

Step 3: evaluate performance

- Confusion matrix: predictions against true labels

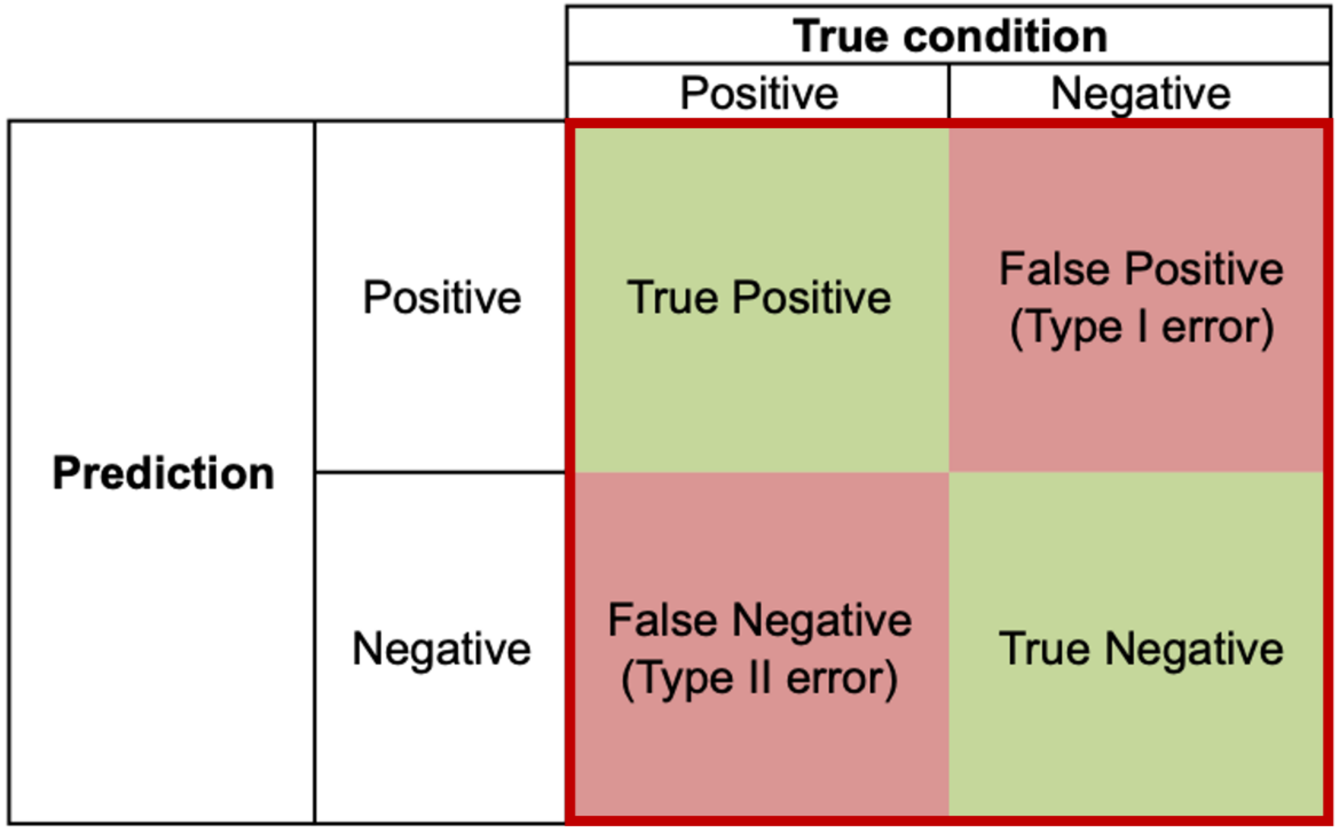


		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Overview of Process

Step 3: evaluate performance

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$

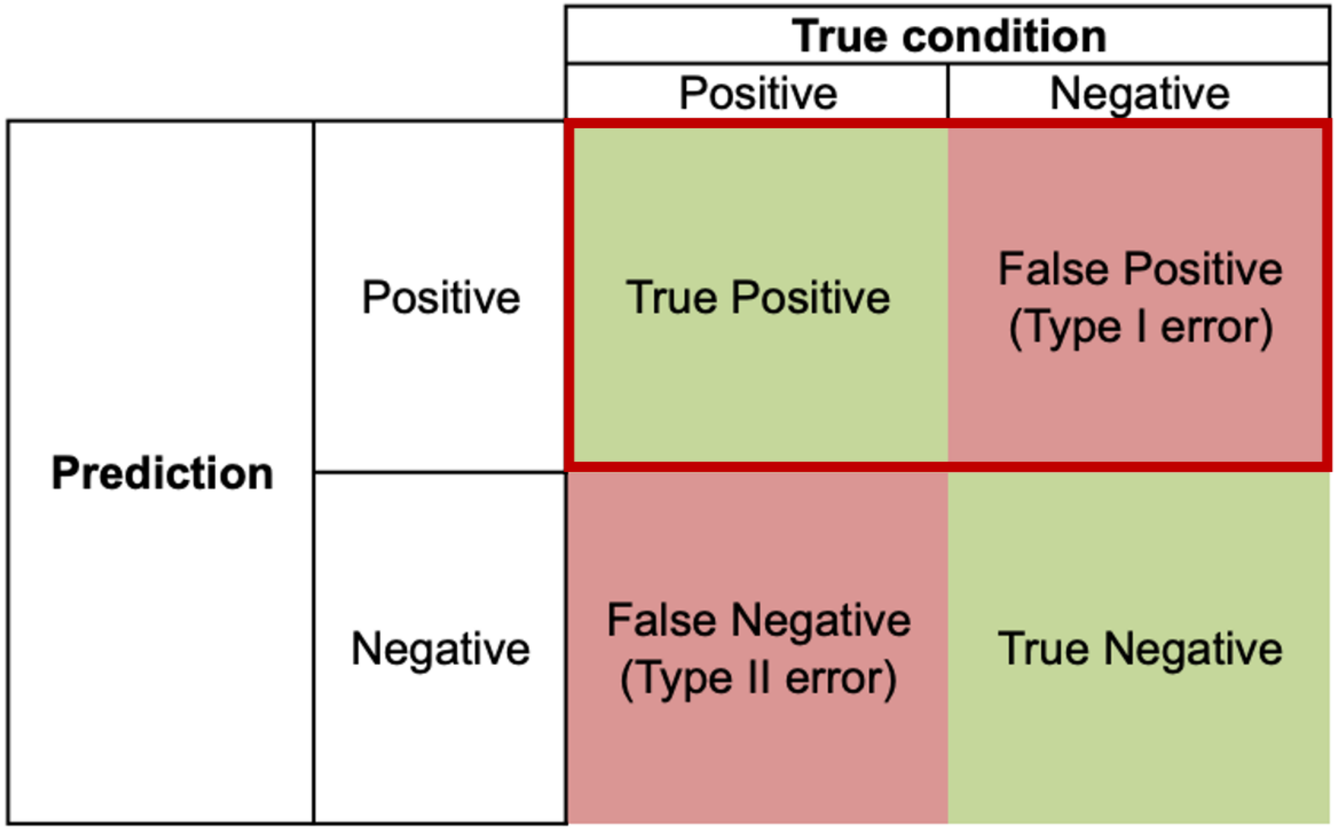


		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Overview of Process

Step 3: evaluate performance

- Precision: $\frac{TP}{TP+FP}$

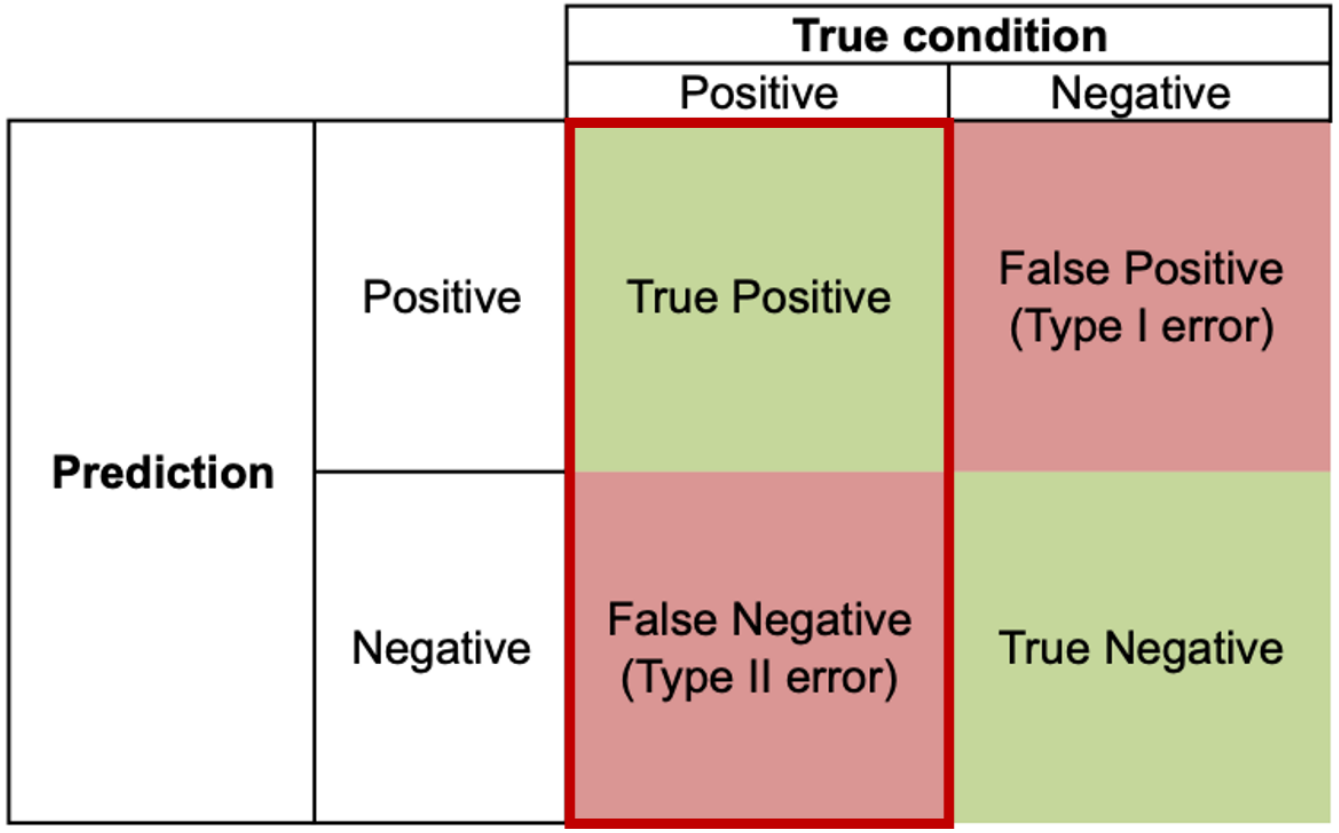


		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Overview of Process

Step 3: evaluate performance

- Recall: $\frac{TP}{TP+FN}$

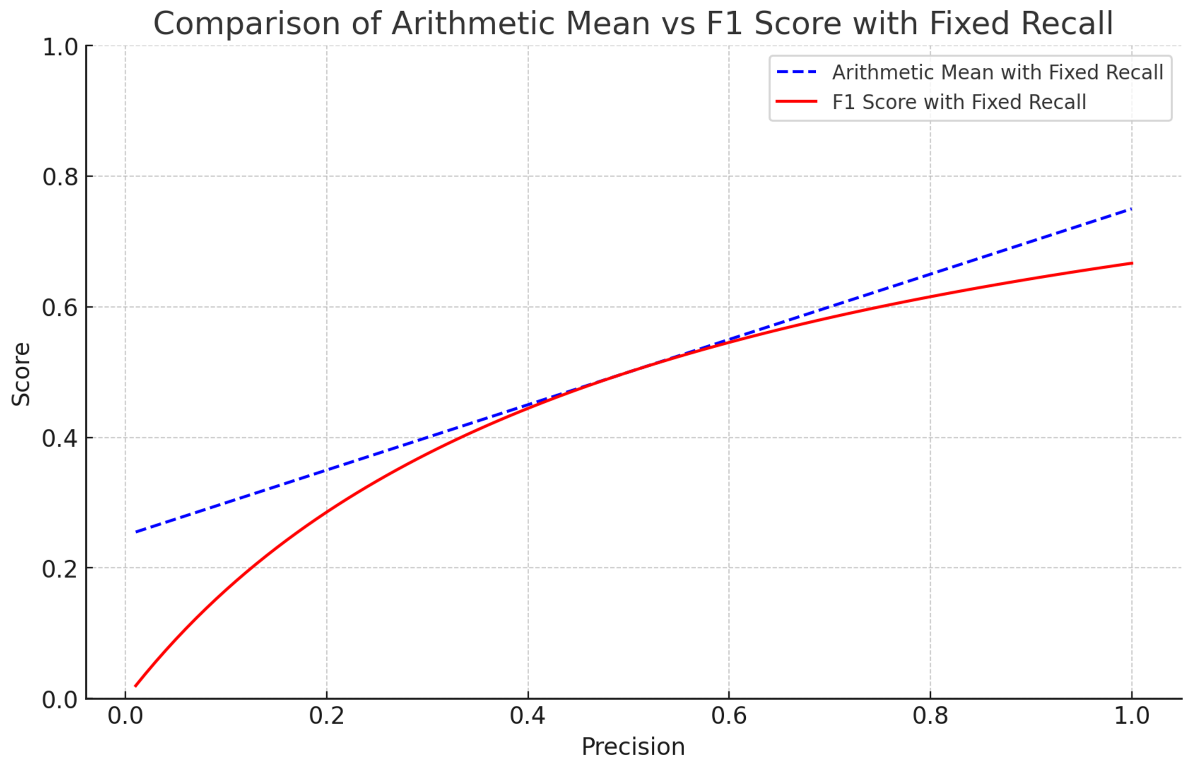


		True condition	
		Positive	Negative
Prediction	Positive	True Positive	False Positive (Type I error)
	Negative	False Negative (Type II error)	True Negative

Overview of Process

Step 3: evaluate performance

- F-1: $(2 \times precision \times recall)$ / $(precision + recall)$
- Why not arithmetic mean ($(precision + recall)/2$)?



Fine-tuning BERT

What is fine-tuning, and why?

- Pre-trained language models (like BERT) capture general, transferable knowledge from massive text corpora
- This knowledge can be adapted for a wide variety of downstream tasks (e.g., classification, QA, NER)
- Fine-tuning adds task-specific layers and updates the model using supervised data from the target task

→ Stronger performance than training a new model from scratch, especially with limited data

Fine-tuning BERT

Overall process of fine-tuning BERT

- `[CLS]` token represents the entire input sequence
- It is a special token added to the start of input during fine-tuning (also during pre-training)
- An vector is learned for this token (a.k.a. “sentence embedding” since it refers to the entire sequence)
- The output vector in the final layer for `[CLS]` serves as representation of the entire input sequence (a summary representation)

Fine-tuning BERT

Overall process of fine-tuning BERT

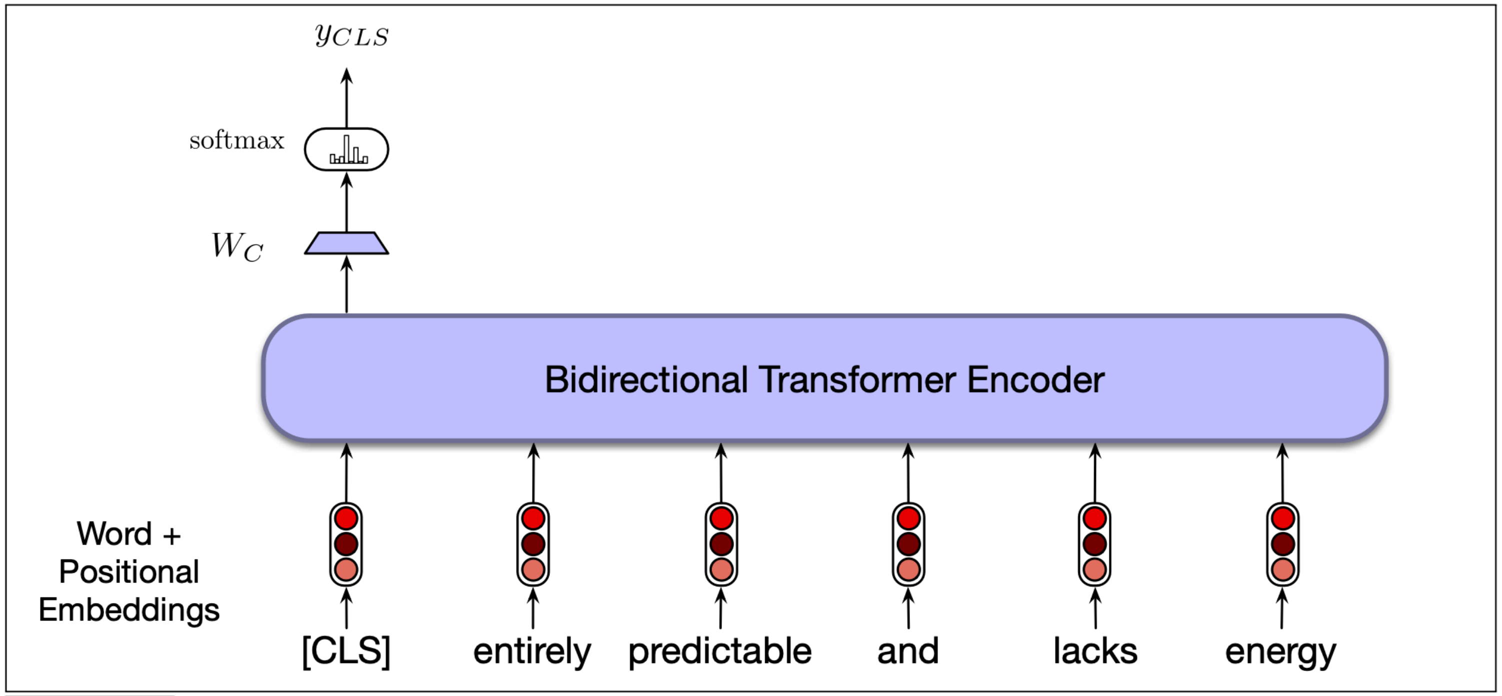

y_{CLS}
softmax
W_C
Bidirectional Transformer Encoder
Word + Positional Embeddings
[CLS]
entirely
predictable
and
lacks
energy

Figure 11.8 Sequence classification with a bidirectional transformer encoder. The output vector for the `[CLS]` token serves as input to a simple classifier.

Generative LLMs

What do we mean by LLMs?

- Overall, LLMs tend to refer to generative models
  - GPT, LLaMA, Claude, Gemini, etc.
- Various terminology: autoregrsesive, left-to-right, or causal models
- Autoregressive models
  - Use their earlier predictions to make later predictions (e.g., the model’s first generated token is used to generate the second token)
- They are primarily decoder-only models
  $\leftrightarrow$ encoder-only (representation) models (e.g., BERT)
  $\leftrightarrow$ encoder-decoder (seq-to-seq) models (e.g., T5, BART)

Generative LLMs

Autoregressive models do not generate all at once

- They generate one token at a time (just like ChatGPT)
- Each token generation step is one forward pass through the model
    - The input tokens go into the model and flow through the computations to produce an output
- After each token generation, the generated token is appended to the end of the original input from the previous run
- So the model is run in a loop to sequentially expand the generated text until completion

Generative LLMs

Autoregressive models do not generate all at once

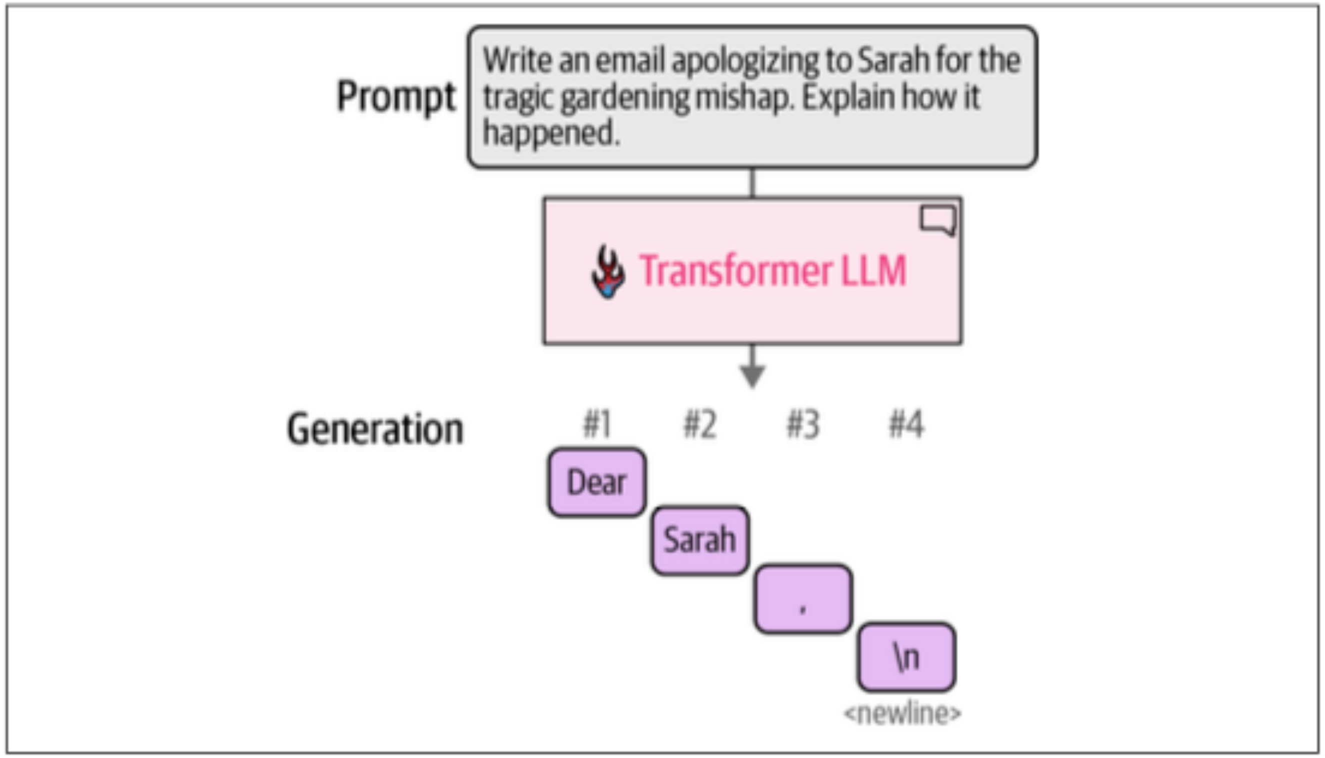
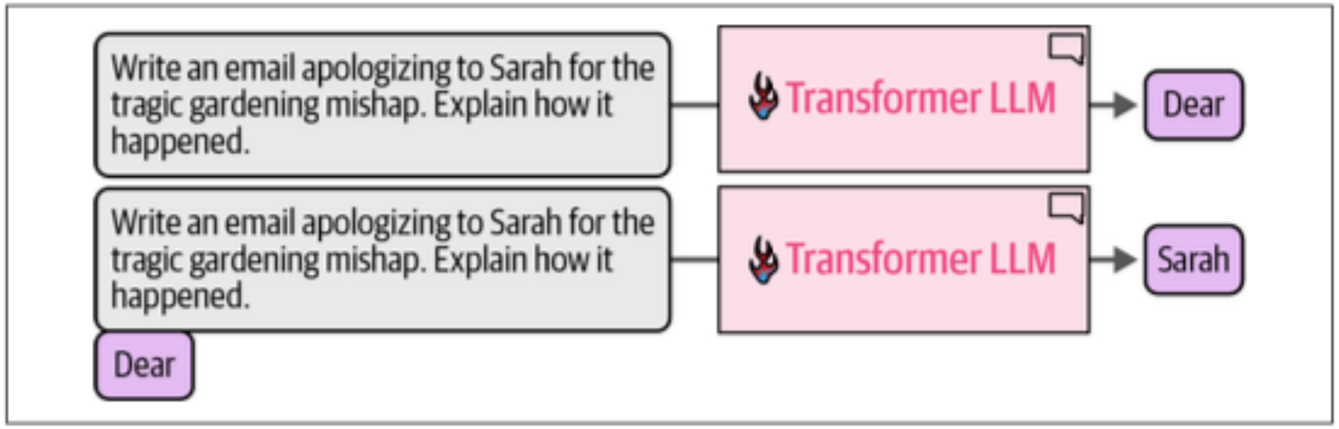

Prompt
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Generation
#1
#2
#3
#4
Dear
Sarah
,
\n
<newline>

Figure 3-2. Transformer LLMs generate one token at a time, not the entire text at once.

Generative LLMs

Autoregressive models do not generate all at once



Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Dear
Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Dear
Transformer LLM
Sarah

Figure 3-3. An output token is appended to the prompt, then this new text is presented to the model again for another forward pass to generate the next token.

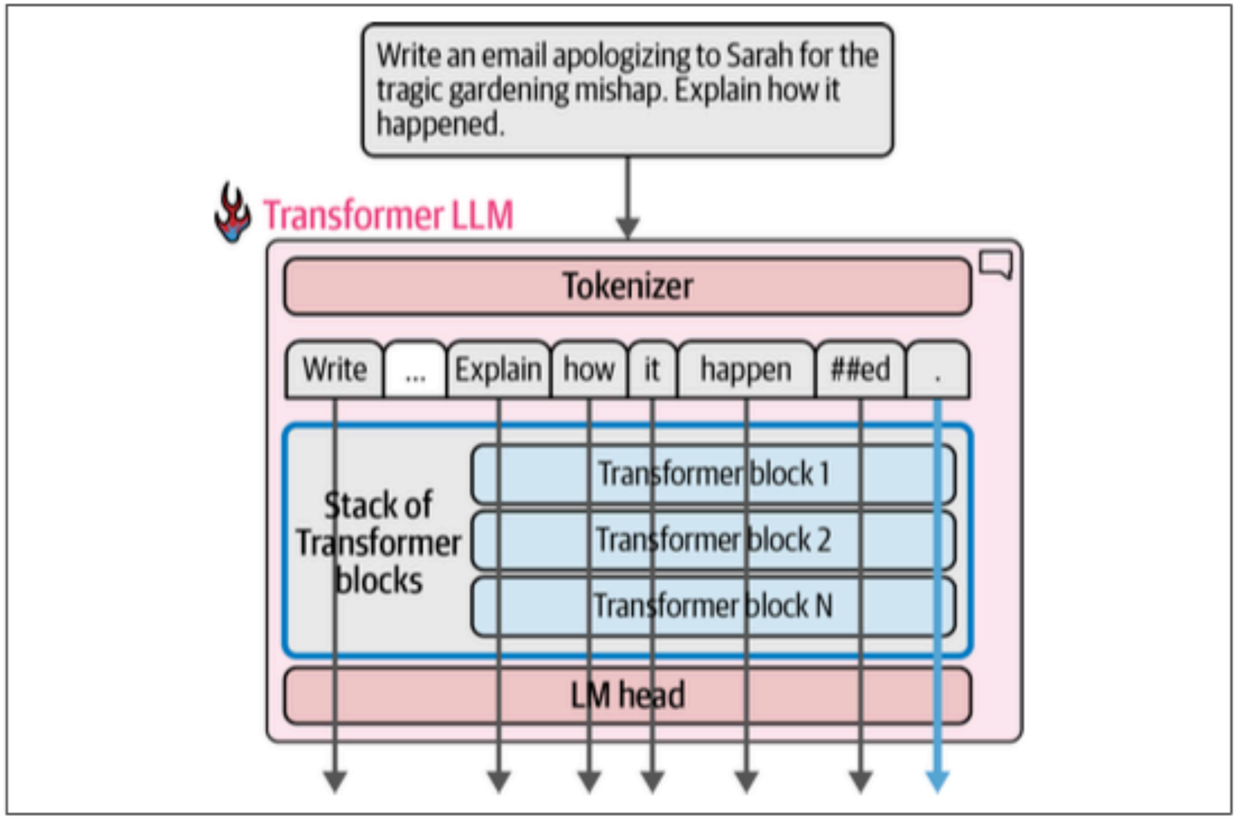
Generative LLMs

How is the next token predicted?

- Only the final token’s output vector (= vector representation) is passed into the final layer (language model head) to predict the next token
- This head produces a probability distribution over the vocabulary
- Then, why compute representations for all tokens if we discard all but the last?
  - The final output vector incorporates contextual information from all preceding tokens
  - This allows the vector to reflect the overall meaning of the input

Generative LLMs

How is the next token predicted?



Write an email apologizing to Sarah for the tragic gardening mishap. Explain how it happened.
Transformer LLM
Tokenizer
Write
...
Explain
how
it
happen
##ed
.
Stack of Transformer blocks
Transformer block 1
Transformer block 2
Transformer block N
LM head

Figure 3-8. Each token is processed through its own stream of computation (with some interaction between them in attention steps, as we'll later see).

Generative LLMs

How is the next token predicted?

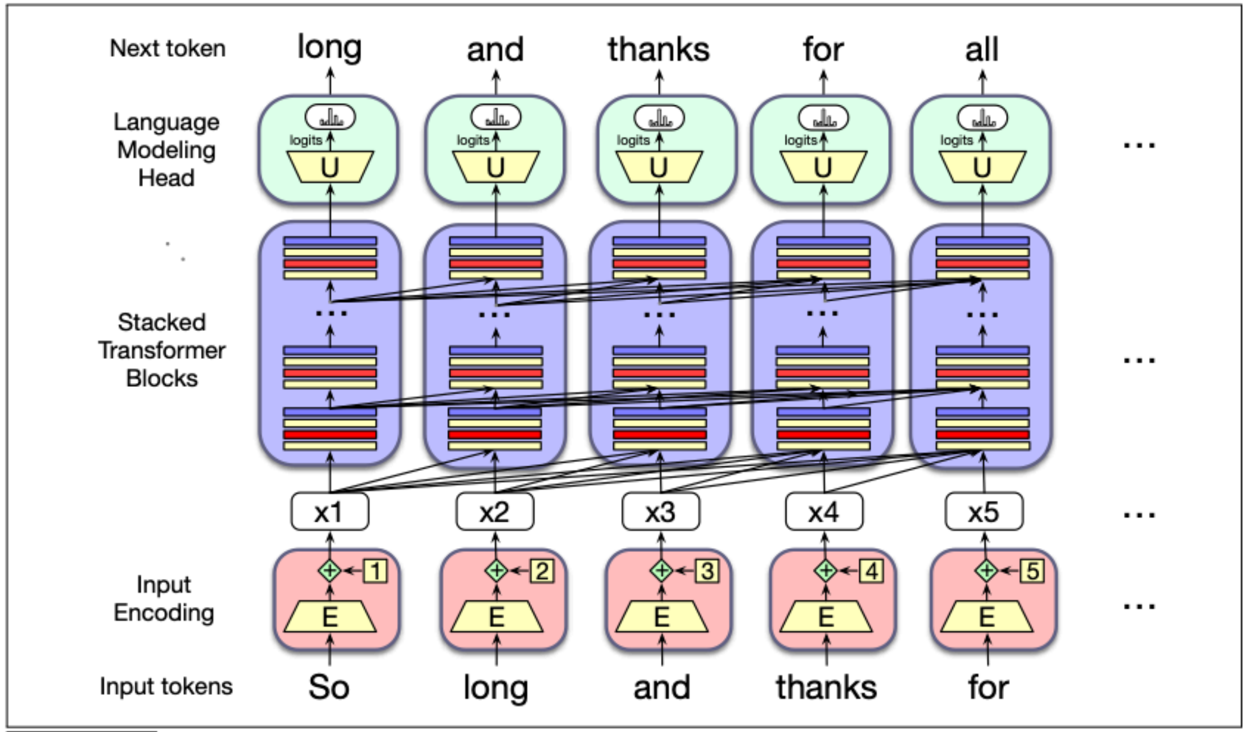


Figure 9.1 The architecture of a (left-to-right) transformer, showing how each input token get encoded, passed through a set of stacked transformer blocks, and then a language model head that predicts the next token.

Language Modeling Head

Once we have a vector representation for the last token

- This should be mapped to a probability distribution over $|V|$
- The task of the language modeling head is to take the output of the final transformer layer from the last position and use it to predict the upcoming word at the next position
- It takes the output of the last token at the last layers and produces a probability distribution over $|V|$
- Details can be found pp. 16–18 [JM]

Language Modeling Head

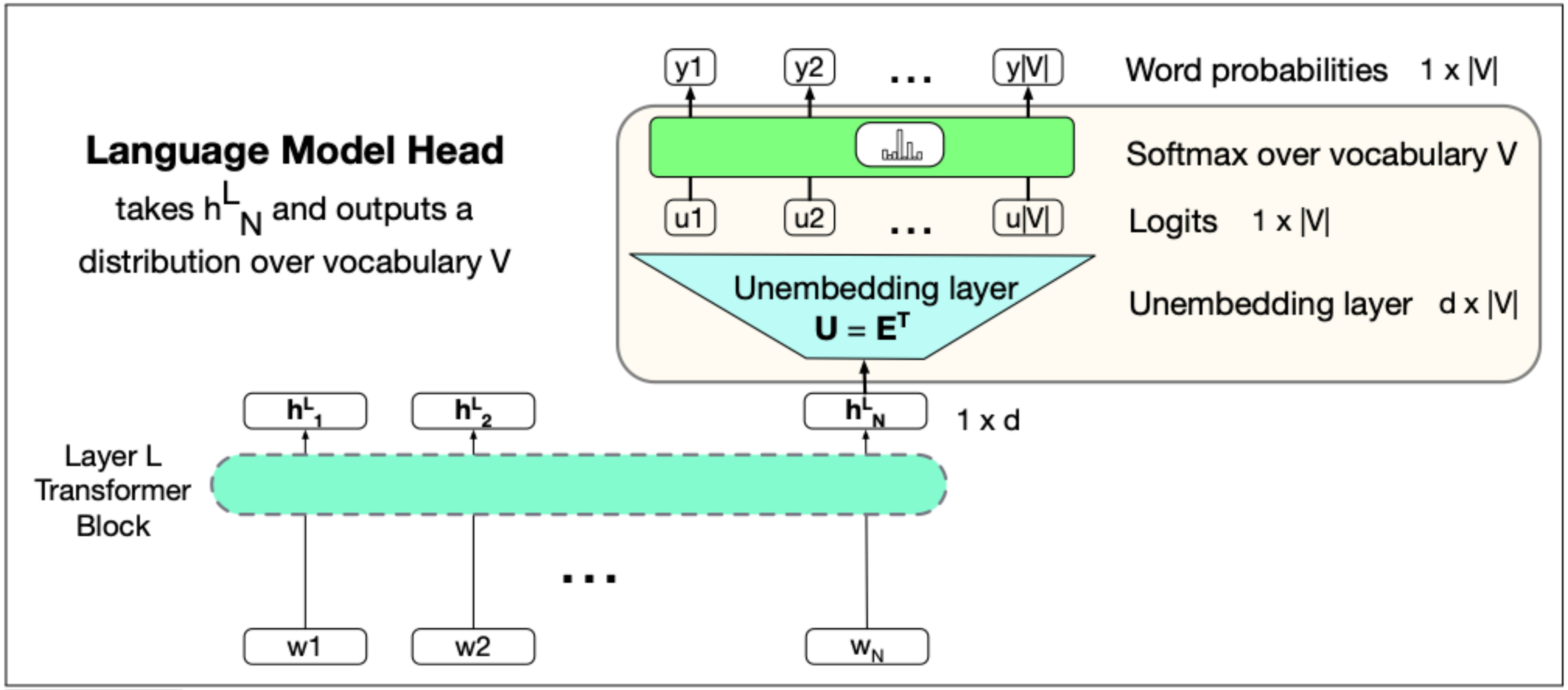

Language Model Head
takes h^L_N and outputs a distribution over vocabulary V
y1
y2
...
y|V|
Word probabilities 1 x |V|
Softmax over vocabulary V
u1
u2
...
u|V|
Logits 1 x |V|
Unembedding layer
U = E^T
Unembedding layer d x |V|
h^L_1
h^L_2
h^L_N
1 x d
Layer L Transformer Block
...
w1
w2
w_N

Figure 9.14 The language modeling head: the circuit at the top of a transformer that maps from the output embedding for token N from the last transformer layer ($\mathbf{h}^L_N$) to a probability distribution over words in the vocabulary V.

Decoding

The task of choosing a word to generate based on the model-generated probabilities

- Greedy decoding
  - Always selects the token with the highest probability at each step
    - Simple and fast, but can lead to bland outputs
    - Deterministic: given the same prompt, always produces the same output
- Sampling-based decoding
  - Selects the next token by sampling from the probability distribution
  - Allows for more diverse and creative text generation
  - Stochastic: same prompt may produce different outputs each time

Decoding

Varieties of sampling-based decoding

- Pure/random sampling
  - Sample directly from the probability distribution
  - *How can this go wrong?*
- Top-k sampling
  - We truncate the distribution to the most likely tokens, re-normalize to produce a legitimate probability (= adding up to 1), and then randomly sample form within these k words
  - When k=1, top-k sampling = greedy decoding
  - When $k > 1$, it leads to choosing a token that is not necessarily the most probable

Decoding

Varieties of sampling-based decoding (cont'd)

- Top-p sampling
  - Depending on the distribution, the most probable tokens from top-k sampling will only include a fraction in the probability distribution
  - Top-p sampling (a.k.a. nucleus sampling) keeps the top p percent of the probability mass
  - Likely more robust with various shapes of of the probability distribution
  - Higher p yields more randomness

Decoding

Varieties of sampling-based decoding (cont'd)

- Temperature sampling
  - Originates from thermodynamics (a system at a high temperature is flexible)
  - Instead of truncating the distribution, we reshape it
  - For more randomness, we make the distribution flatter
  - Higher temperature parameter, τ, indicates more randomness ($\tau = 0$: greedy decoding)

Generative LLMs for Text Classification

Promises

- No need for labeled training data (zero-shot/few-shot prompting)
- High performance in various tasks (classification, sentiment, ideology)
- Little programming or machine learning expertise required
- Multi-lingual and adaptable across domains

Generative LLMs for Text Classification

Pitfalls

- Transparency and replicability issues with closed models (e.g., GPT-4)
- Computational power (open-source models) or API usage fees (closed-source, proprietary models)
- Lack of validation framework: $\leftrightarrow$ Traditional ML tools (e.g., train-test splits, CV) do not directly apply
- Small prompt changes can produce drastically different results

Generative LLMs for Text Classification

Recommended pipeline (Törnberg et al. 2024)

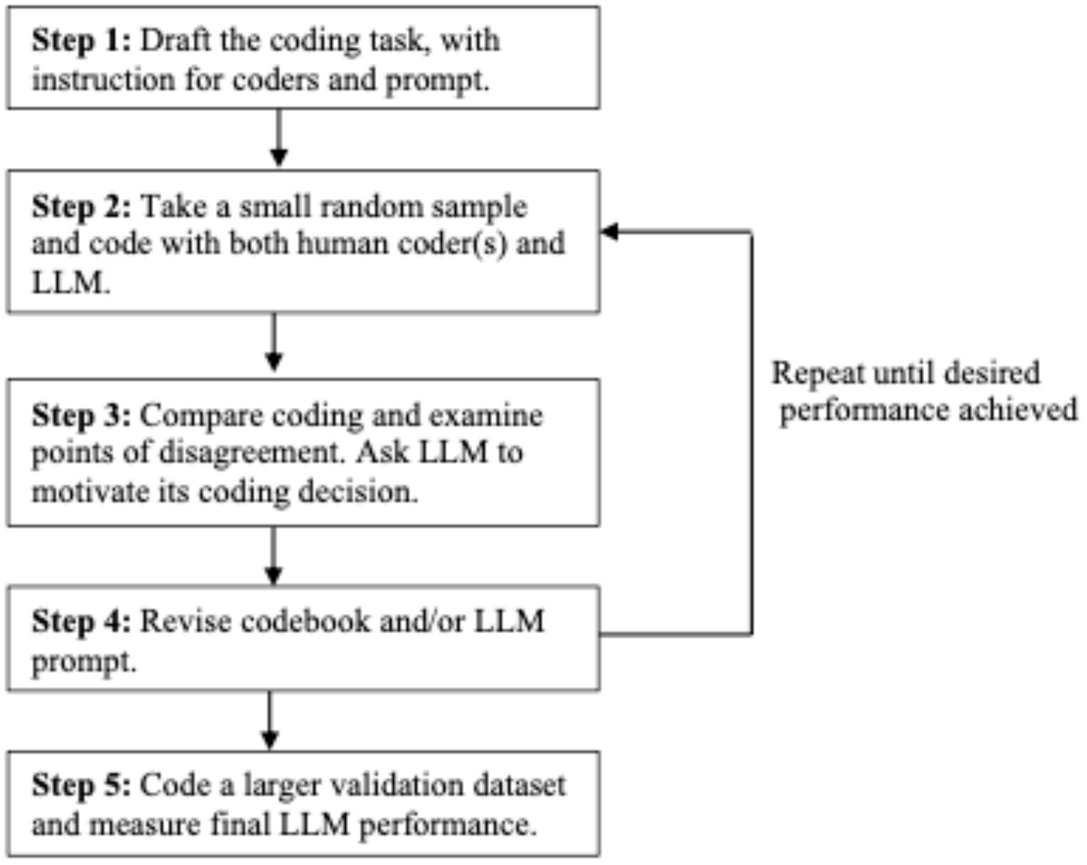

Step 1: Draft the coding task, with instruction for coders and prompt.
Step 2: Take a small random sample and code with both human coder(s) and LLM.
Step 3: Compare coding and examine points of disagreement. Ask LLM to motivate its coding decision.
Step 4: Revise codebook and/or LLM prompt.
Step 5: Code a larger validation dataset and measure final LLM performance.
Repeat until desired performance achieved

Figure 1: Example of a systematic coding procedure.

Guided Coding

- Stance detection on abortion with BERT fine-tuning
- Introduction to text classification with generative models
- Using OpenRouter for text classification