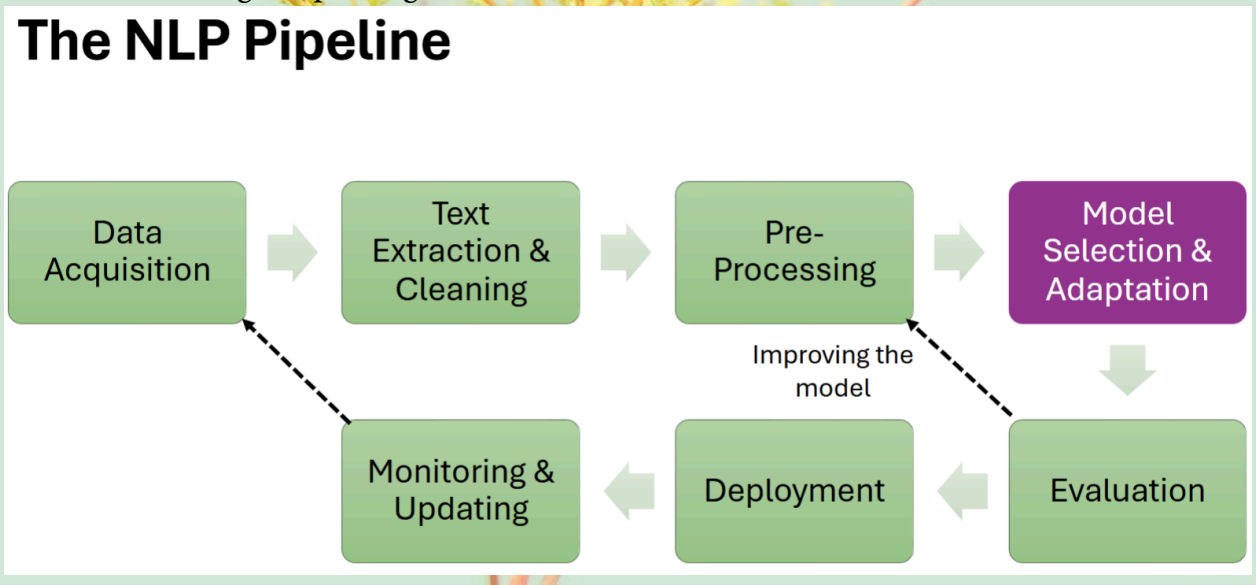
NLP pipeline

Into NLP - Fall 2025

NLP PIPELINE

MODERN APPROACH - pretrained Large Language Models & adapt them to specific needs. Emphasizes model adaptation & intelligent prompting.
LLM correction: Context aware fixes
Traditional tools: Microsoft REST API, custom dictionaries
Hybrid approach: Combo of both
OLD APPROACH -

1. Data acquisition
2. text extraction - optional
3. pre-processing - optional
4. model adaptation
5. evaluation
6. deployment
7. monitoring & updating -



Data acquisition

Data can come from anywhere. Can visit how huggingFace collects data

Ideal scenario - labeled data. Hardest to find.
Limited scenario - artificially generated data or just using few-shot training to reduce need for data.
Sources for data - from social media platforms, product support teams, etc.
Corpus: large collection of text/audio data for training NLP
Requirements - Ethics, Quality, Quantity, and Variety
Risks - Privacy, policy conflicts, permissions (no explicit consent from user), copyright
Artificial data - LLM-based generation, auto-labeling, data augmentation (rewording of data), active learning (iteratively learn data to maximize learning efficiency), and synthetic data creation.
Back translation: Translate English language to another language then translate it back to English (some languages may phrase things differently)

text extraction

Spell checks
Converting to unicode characters

Pre-processing

Modern consideration - Transformers can automate many preprocessing steps.

- Remove unnecessary text
- Structure unstructured text
- Clean the data

<u>Common steps</u>

- Preliminaries - sentence segmentation, word tokenization
- Frequent steps - stop word removal, stemming, lemmatization, removing digits, lowercasing
- Advanced processing - Part of Speech (POS) tagging, parsing, conference resolution
- Other steps - sentence segmentation, lemmatization(eating, ate, eats all converted to “eat”)

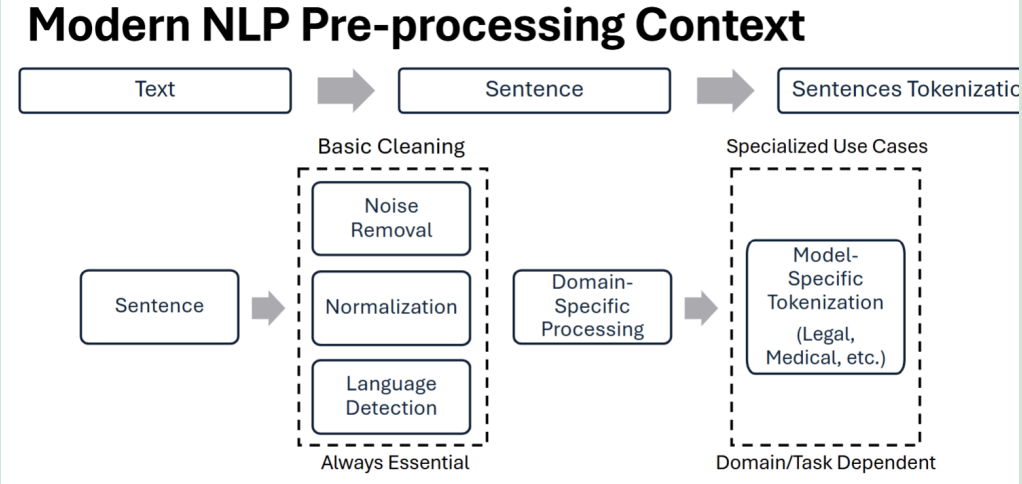
Advanced processing

Text	Lemma	POS	Shape	Alpha	Stop
Missouri	Missouri	PROPN	Xxxxx	True	False
is	be	AUX	xx	True	True
known	know	VERB	xxxx	True	False
for	for	ADP	xxx	True	True
its	its	PRON	xxx	True	True
beautiful	beautiful	ADJ	xxxx	True	False
rivers	river	NOUN	xxxx	True	False
and	and	CCONJ	xxx	True	True
vibrant	vibrant	ADJ	xxxx	True	False
cities	city	NOUN	xxxx	True	False
.	.	PUNCT	.	False	False
It	it	PRON	Xx	True	True
became	become	VERB	xxxx	True	True
a	a	DET	x	True	True
state	state	NOUN	xxxx	True	False
in	in	ADP	xx	True	True
1821	1821	NUM	dddd	False	False
.	.	PUNCT	.	False	False

One common challenge is maintaining the relationships between words when converted to embeddings.

modern context

Basic cleaning: Removing noise & normalization
Tokenization: Now handled by specific models such as GPT/BERT
Advanced linguistics: Often learned automatically but useful for specialized domains



Traditional feature engineering

Part of Model Selection & Adaptation step

3 evolutionary approaches

1. Classical NLP: Manual feature engineering.

- explicit design required by hand
- distinct phase between pre-processing & modeling
- sparse representations - high-dimensional mostly zero vectors.
  Bag-of-words: word occurrence counting
  TF-IDF: Term frequency weighted by inverse document frequency
  N-gram features: Sequences of consecutive words
  linguistic features: POS tags, NER, syntactic patterns
  Statistical features: Text length, readability scores
  Feature selection: Manually remove irrelevant features

Traditional workflow:

Documents -> preprocessing -> manual feature extraction -> classical ML -> Output

2. Deep Learning NLP (2017-2020): Automatic feature learning

Automatic feature learning: Neural networks discovered patterns in data
Dense embeddings: Pre-trained representations. Ex:

- **Word2Vec/GloVe embeddings**: Words as dense vectors to capture semantic similarity.

End-to-end training: Features learned at training time & not determined beforehand
Hidden layer representations: Multi-levels of abstraction
Doc2Vec: Document-level representations.
CNN/RNN architecture: Auto pattern-detection in text
Transfer learning emergence: Pre-trained embeddings for multiple tasks
Transformer architecture: Self-attention mechanisms
Parallel processing: Faster to train than RNNs

Deep learning workflow:

Documents -> preprocessing -> pre-trained embeddings-> neural network -> Output

3. Modern Transformer Era: Pre-trained model adaptation

NO MANUAL FEATURE ENGINEERING

Pre-trained transformers: BERT, GPT, RoBERTa learns all features automatically
Contextual embeddings: Dynamic representations based on surrounding context
Parameter-efficient fine-tuning: LoRA/QLoRA adapt models without feature design
Prompt engineering: Natural language instructions replace feature crafting
BERT: Bidirectional training understanding context from both directions

Pre-trained foundation models

- Good for understanding text - DeBERTa, RoBERTa, BERT
- Good for generating text - T5, BART
- LLM - GPT-4o, Claude, gpt-oss, etc

Finetuning methods

- **LLM Prompting** - Best for complex reasoning, limited data, rapid prototyping
    - **0-shot**: Task description + input->output
    - **few-shot**: Task description + examples + input->output
    - **chain-of-thought**: Encourage step-by-step reasoning
- **Parameter-efficient fine-tuning** - Task-specific optimization, resource constraints.
    - **LoRA (Low Ranking-Adaptation)**: 90-95% fewer parameters to train
    - **QLoRA (Quantized LoRA)**: 4x memory reduction through quantization
- **Cloud APIs** - Production deployment, scaling, compliance
    - **Google Cloud NLP**: Integrated ecosystem
    - **OpenAI API**: Industry standard LLMs
    - **Azure OpenAI**: Enterprise deployment

WHAT REPLACED TRADITIONAL FEATURE ENGINEERING?

Transformer attention mechanisms: Auto-identify important text patterns
Contextual understanding: Same word has different representations in different contexts
Multi-layer representations: Deep hierarchical feature learning
Transfer learning at scale: Models trained on billions of tokens

Modern workflow:

Documents -> Minimal preprocessing -> pre-trained adaptation -> Output

Aspect	Classical NLP (Pre-2017)	Deep Learning NLP (2017-2020)	Modern Transformers (2020-2025)
Feature Design	Manual, expert-driven	Semi-automatic, learned	Fully automatic, contextual
Representation	Sparse vectors (TF-IDF)	Dense embeddings (Word2Vec)	Contextual embeddings (BERT)
Engineering Effort	High - months of feature design	Medium - embedding selection	Low - prompt/fine-tuning
Performance	Limited by feature quality	Better with learned features	SOTA with pre-trained models
Interpretability	High - features are explicit	Medium - some interpretability	Variable - attention visualization
Data Requirements	<10K examples sufficient	10K-100K examples needed	100-1K examples with fine-tuning
Computational Cost	Low - simple algorithms	High - neural network training	Variable - API vs. self-hosting

Modeling

Evaluation

Metric	Type	Description	Advantages	Best Use Cases
BERTScore	Semantic	Contextual embedding similarity using BERT	- Better human correlation - Context-aware - Multilingual support	Text generation, summarization, translation quality
LLM-as-Judge	AI-powered	GPT-4/Claude rates outputs on multiple criteria	- Human-like judgment - Multi-dimensional - Scalable	Complex reasoning, creativity, helpfulness assessment
BLEURT	Learned	BERT-based learned evaluation metric	- Robust to paraphrasing - Trained on human ratings	Machine translation, text generation
G-Eval	Generated	LLM generates evaluation criteria and scores	- Task-adaptive - Explainable reasoning	Custom evaluation scenarios
Traditional F1	Statistical	Harmonic mean of precision and recall	- Interpretable - Fast computation - Established baseline	Classification, NER, simple tasks
ROUGE-L	N-gram	Longest common subsequence overlap	- Captures sentence structure - Order-sensitive	Summarization (legacy comparison)
Perplexity	Probabilistic	Model uncertainty/confusion measure	- Language model quality - Generation fluency	Language modeling, text generation fluency

Deployment, monitoring & updating

Deployment Type	Best For	Advantages	Tools & Platforms
Cloud APIs	Rapid deployment, variable load	No infrastructure management, instant scaling	OpenAI API, Azure OpenAI, AWS Bedrock
Containerized Self-Hosting	Custom control, consistent performance	Full control, cost predictable	Docker + Kubernetes, TorchServe, TensorFlow Serving
Edge Deployment	Low latency, offline capability	Reduced latency, privacy-preserving	TensorFlow Lite, ONNX Runtime, Tiny Language Models
Serverless Functions	Event-driven, sporadic usage	Pay-per-use, automatic scaling	AWS Lambda, Google Cloud Functions, Azure Functions
CI/CD Integrated	Continuous deployment	Automated testing and deployment	GitHub Actions, GitLab CI/CD, Jenkins

Processing text

Intro to NLP - Fall 2025

Text representation

Machines understand digits better than text because text can be represented by digits.

- Images - represented as matrix of pixel intensities
- Video - sequential collection of image matrices (frames)
- Speech - Sample sound wave at fixed intervals. Record amplitude at each point.

Text representation must capture:

- lexical units (words, phrases)
- meaning of each unit
- syntactic structure
- context in a sentence

Basic vectorization approaches

Vector Space Models (VSM): Convert text to mathematical vectors represented in high dimensional space. For information retrieval, document classification, and clustering

- **cosine similarity**: cos(θ)

One-hot encoding

One-hot encoding: Unique word/category assigned vector with 1 "1" and all other positions "0". Ex: "cat chases mouse" so cat = [1,0,0], chases = [0,1,0], mouse = [0,0,1]

- **high dimensionality** - vector size equals vocabulary size
- **sparsity** - most entries are 0s, so storage and computation
- **Out-of-vocabulary (OOV) problem** - new words can't be represented without retraining

Bag of Words

Bag of Words (BoW): Ignore input's ordering & structuring, focusing only on words' occurrences. Helps understand which words influence predictions. Efficient for small datasets. Ex: "Hello, who are you? Hello?" = {"Hello": 2, "who": 1, "are": 1, "you": 1}

1. **tokenization**: Split each text into individual words/tokens. Split at whitespace/punctuations

2. Vocabulary building
3. **Encoding**: Count how often each word from vocabulary appears. Resulting vector is word frequencies for all given text.

Pros:
No training needed unlike embeddings. Simple code to create these vectors. Good for document classification. Interpretable results.
Cons:
Stopwords occur frequently but aren't important. No word order. Most entries are still 0. No semantic understanding. Vocabulary size can be too big.

TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency): Used weighting scheme to highlight important (rare) words in documents. Reduces impact of common words. Widely used for information retrieval & classification. Used in search engines.

- Formula for Term Frequency = # of word's occurrence in text/total # of words in text = probability
- Formula for Inverse Document Frequency = $idf(t) = log(\frac{1+N}{1+df(t)}) + 1$. Where N = total number of documents. $df(t)$ = # of documents containing term t.
- Formula for TF-IDF = tf(t,d) * idf(t)
- Formula for L2 normalization = Divide by its Euclidean norm = norm(V) = $\sqrt{\sum_i v_i^2}$

Pros - Efficient, no training needed. Interpretable to see which term drives the prediction.
Cons - Inconsiderate of word combinations. Ex: "get out" can better capture meaning then separately "get" and "out".

Bag of N-grams

Bag of N-grams: Better preserves word order & context than BoW.
Pros - Fast & interpretable. No training data needed. Good for document classification. Handles domain-specific terms well
Cons - Vocabulary size may grow large because for example: "get out" AND "get" both need to be stored. Not every combination makes sense

Word embeddings

Distributed representations

Traditional vectorization methods have disadvantages such as high dimensionality and sparsity. Distributed representations such as ones used in neural networks learn through dense, low-dimensionality representation of words & texts.

- **Distributional Similarity**: Word's meaning is inferred from context.
- **Distributional Hypothesis**: Words appearing in similar contexts have similar meanings. Ex: "dog" and "cat"
- **Distributional representation**: High-dimensional, sparse vectors derived from word co-occurence in context. Ex: One-hot encoding, bag of words, n-grams, TF-IDF.
- **Distributed representation**: Low-dimensional, dense vectors that compress distributional representations for computational
- **embedding**: Compresses high dimensional distribution space to low dimensional distribution space
- **vector semantics**: NLP methods that learn word representation based on distributions in large corpora

Feature	Distributional Representation	Distributed Representation
Dimensionality	High	Low
Sparsity	Sparse	Dense
Example Methods	One-hot, Bag of Words, TF-IDF	Word Embeddings (Word2Vec, GloVe)
Computational Efficiency	Low	High

Word2Vec

Word embedding: Maps words to dense numerical vectors in high-dimensional space while capturing semantic relations between words. Words with similar meanings are closer together this lets machines process language more effectively.

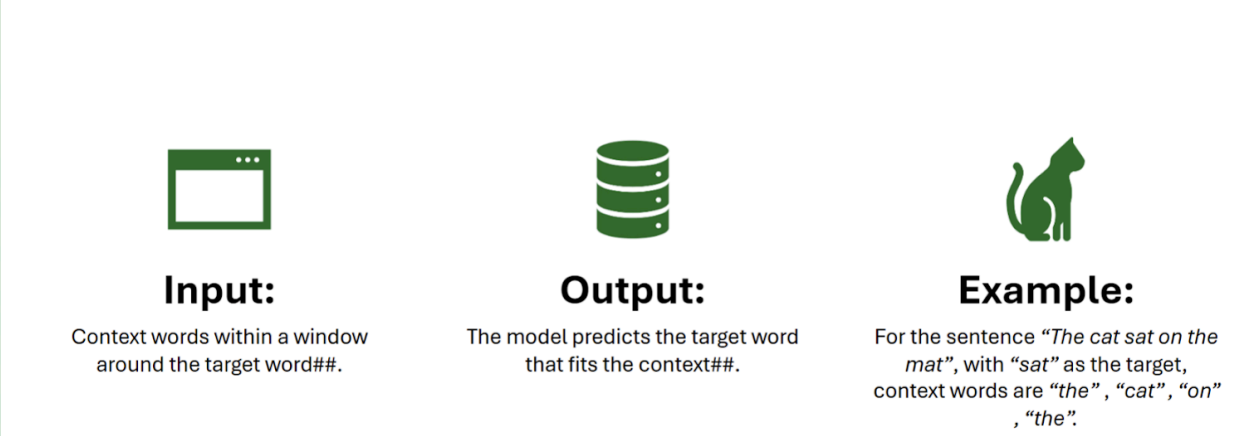
Usages:

- text classification
- named entity recognition
- information retrieval
- machine translation

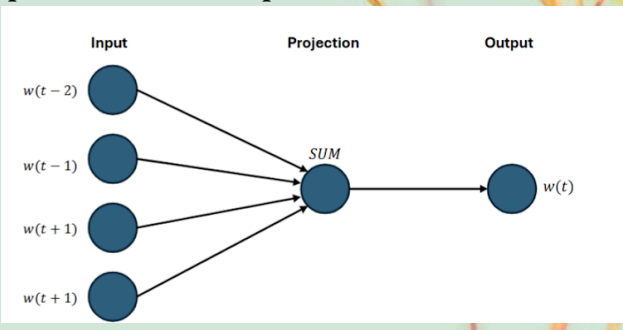
- recommendation system

Word2Vec: Neural network based model for learning word embeddings developed by Google in 2013. First practical dense embeddings that made semantic similarity computable. Foundations for BERT, GPT, and transformers.

CBOW model



Predict a target word using surrounding words.
Window: How many words model can see at once.
Input layer: Represents context words as 1-hot vectors. Projected into embedding space & averaged to form context vector. Input is 1-hot vector for target word
Hidden layer: Projects this into embedding space.
Output layer: Uses softmax to predict probability of each word in vocabulary as target. (probabilities of all possible context words)



Skip-gram model

Predict surrounding words using a target word.
Input - current (target) word.
Output - Model predicts each context words
Ex: For "sat" as the target, predicts "the," "cat," "on," "the" as context words.

Negative sampling: For each positive (correct) word pair, a few negative (incorrect) pairs are sampled & used to update model. Speeds up training & improves embedding quality.

Evaluating word similarity

cosine similarity: Similarity ranges from -1 (complete opposite) to 1 (identical)

- When testing similarity scores, adjust:
  - vector_size (embedding dimension)
  - window (context size)
  - min_count (minimum word frequency)

$$\text{cosine similarity} = \frac{\vec{A} \cdot \vec{B}}{||\vec{A}|| \times ||\vec{B}||}$$

Word Pair	CBOW Similarity	Skip-Gram Similarity
'alice' & 'wonderland'	0.9866	0.8666
'alice' & 'machines'	0.9544	0.8534

limitations

static embeddings: "Bank" has same representation in "river bank" and "money bank"
Out-of-vocabulary: Can't handle new words without retraining
No subword information: can't understand "unhappy" from "happy"
Limited context: Fixed window size, no long-range dependencies. (Addressed in modern methods)

BERT (Bidirectional Encoding Representations from Transformers)

Sequential models: Processes words in 1 direction either left to right or right to left. This limits it to only past/future context and not both. Ex: “River” seen after “Missouri” versus “University of” seen before Missouri River.

Aspect	Word2Vec (Previous Lecture)	BERT (Today's Focus)
Representation Type	Static: one vector per word	Contextual: vector changes with context
Context Handling	Ignores surrounding context	Bidirectional: uses full sentence context
Architecture	Shallow neural networks (CBOW/Skip-gram)	Deep Transformer with self-attention
Training Objective	Predict word from context or vice versa	Masked Language Modeling + Next Sentence Prediction
Missouri Example	"Missouri" always same vector	"Missouri" different in "University of Missouri" vs "Missouri River"

Bidirectional: Uses context from left & right directions
Encoder Representations: Uses only encoder part of Transformers
Transformers: Built on Transformer architecture with self-attention
BERT advantages:

- Introduced contextual understanding to NLP
- Led to RoBERTa, DeBERTa, GPT, ChatGPT, Claude
- Production standardized, could be used for search engines, translators, chatbots
- still used

how self-attention works

Every token pays attention to every other token - forest sends strong signal to earlier “Mark” and “Twain”. This moves embeddings towards locational entity.
Contextual weighting occurs at once - “novel” influences “Twain’s” embedding. Shifting it towards an author/person entity.
Result:

- “Twain’s books” -> Twain’s = person
- “Mark Twain park” -> Mark Twain = location

Feature	BERT (Contextual)	Word2Vec (Static)
Context awareness	Yes	No
"Twain" (forest vs. author)	Different vectors	Same vector
Cosine similarity	0.7917	1.0000
Handles Missouri ambiguity	Yes	No

- **BERT:**
  - Captures contextual meaning
  - Differentiates "Twain" in place vs. author roles
- **Word2Vec:**
  - Ignores context
  - Treats all "Twain" uses identically

Text classification

Intro to NLP - Fall 2025

What is Classification?

Classification: Categorizing datapoints into one (binary) or more predetermined classes.

- data can be text, speech, image, video, or numeric

Binary classification: 2 classes.
Multiclass classification: More than 2 classes
Multilabel classification: Each datapoint can have many classes assigned to it.
Extreme classification: Large number of possible labels.
Hierarchical classification: Labels organized in hierarchy, main classification with sub classifications.
Ex: Customer support, content organization, e-commerce, email filtering, language identification, authorship attribution, mental health support, fake news detection.

Text classification pipeline

Real-world projects may not always follow the same linear path, they may be subject to systematic experimentation & refinement

1. Collect & label data
2. Training data
3. Preprocessing & feature extraction
4. train and evaluate classifiers
5. model evaluations (may return to step 3.)
6. deploy & predict new data

1. Collecting & Labeling data

Collect/create labeled dataset for classification task. These are examples models will learn from.

2. Splitting data & choosing metrics

Training set: Used to train the model

Validation set: Used for tuning the model.
Test set: Used for final evaluation
Possible evaluation metrics:

- accuracy
- F1 score
- recall
- precision
- area under ROC curve

3. Feature extraction from text

Transform raw data to feature vectors that machine learning algorithms can process
Possible methods:

- BoW
- TF-IDF
- Word embeddings

Feature engineering can impact model performance

4. Training the Classifier

Train classifier using feature vectors & their labels from training set.
Possible algorithms:

- logistic regression
- support vector machines
- decision trees
- deep learning models

5. model evaluation & benchmarking

Evaluate trained models using chosen metrics on test set. Benchmarks helps compare different models, feature sets, and hyperparameters. Ex: Confusion matrices to show model performance strengths & weaknesses.

6. Deployment & Monitoring

Monitor model's performance in production. Use Key Performance Indicators (KPIs) to determine if results have business value.

Pipelines aren't always necessary...

- Business just want a product that works
- Simple tasks that doesn't need pipeline

- Text classification can be rule-based, thus classification pipeline unnecessary

Text classification APIs

- OpenAI API
- Anthropic Claude
- Cohere Classify
- Google Cloud Nature Language
- Azure Cognitive services
- AWS Comprehend

one pipeline, many classifiers

good data

- UCI MAchine Learning repository
- google dataset search
- kaggle
- Hugging Face Datasets

Why multiple classifiers?

No single classification method that works best for all text classification problems. It's an iterative process that may require adaptation to characteristics of each dataset & task.

- Dataset is highly imbalanced

Oversampling: Increase/duplicate the number of samples to match that to the majority-class samples
Undersampling: Select only subset of data from the majority-class samples.

Comparing classifiers

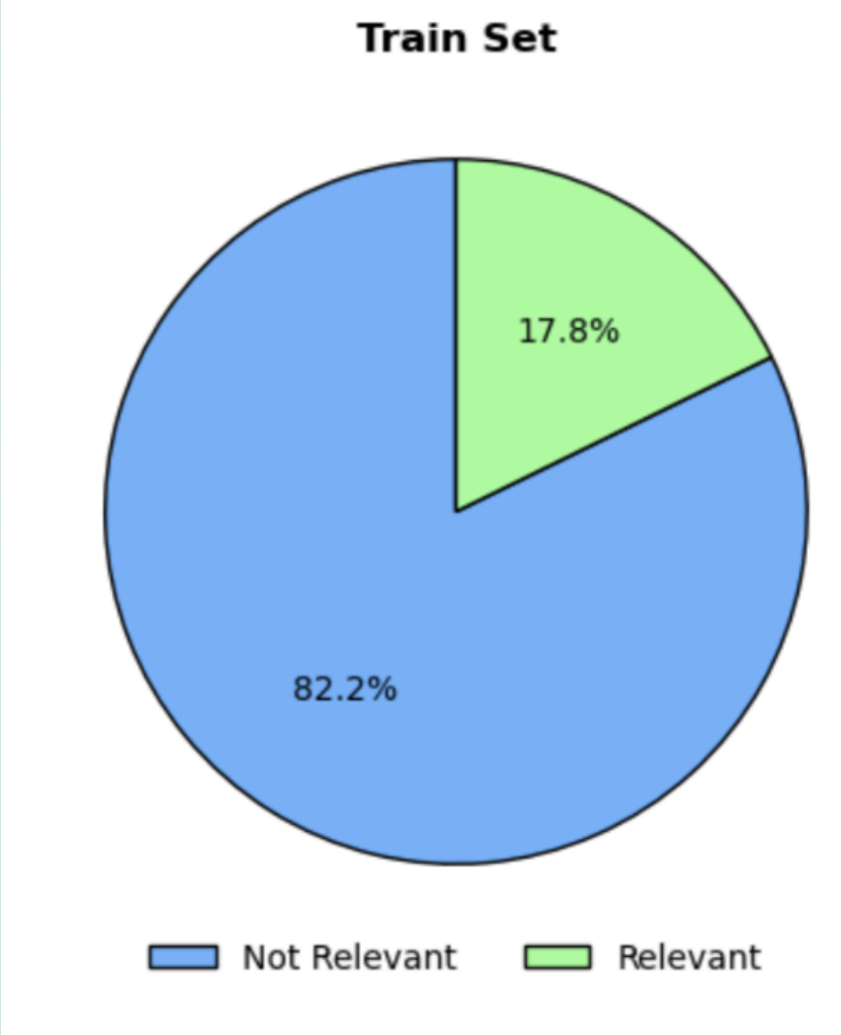
Accuracy alone is misleading, especially with imbalance datasets.
ROC AUC: Better indicator of imbalanced data.

- reducing features (ex: max_features to reduce vocabulary size in BoW) and using class weighting may improve model's focus on minority class and improve recall at cost of accuracy.

Naive Bayes

Naive Bayes: Simple, fast, effective for BoW features

- Assumes features are independent. Each word/feature contributes independently to probability of the class.
- Calculates posterior probability P(A|B) of class A given observed features B (words in text) using Bayes theorem
    - $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$
    - P(A) = prior probability of class A
    - P(B|A) = likelihood to observe B given A
    - P(B) = overall probability of features B (normalizing constant)

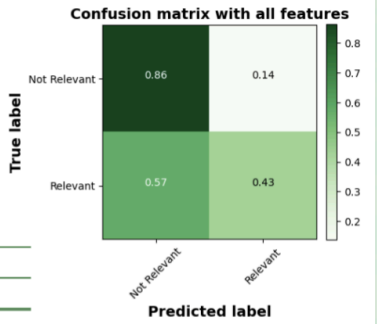


Non-relevant articles:	Correctly identified 86% of the time
	Only 14% misclassification rate
Relevant articles:	Correctly identified only 42% of the time
	58% misclassification rate

Key insight: Model heavily biased toward majority class due to data imbalance

Potential reasons for poor classifier performance

Feature Sparsity	40,000+ features created excessive noise
	Most features appeared in <0.1% of documents
Class	80% non-relevant vs. 20% relevant articles



Accuracy: 78.58% - This appears high, but does not mean the classification was good!
ROC AUC: 74.71% - More accurate. So is F1-score/confusion matrix

LOGISTIC REGRESSION

Logistic regression: Probabilistic outputs. Robust to correlated features

- Create linear boundaries to separate data into classes.
- Discriminative classifier
- Uses logistic (sigmoid) function that outputs probabilities between 0 & 1 for each class
- Can handle large sparse feature spaces like from BoW or TF-IDF
- **class weighting**: penalize misclassification of minority class more heavily. This allows more equitable performance across classes without manual sampling.

$$weight(class_i) = \frac{n_{samples}}{(n_{classes} * count(class_i))}$$

SUPPORT VECTOR MACHINES (SVM)

Support Vector Machines (SVM): Effective for high-dimensional dataset. Can handle non-linear boundaries with kernel tricks.

- Discriminative classifier. Finds hyperplane that maximizes margin between different classes in high-dimensional space.
- Calculates probability of being in 1 class or another.

Limited data size and resources means that Naive Bayes

Neural Word Embeddings

- Document classification
- sentiment analysis

- Machine translation
- Q&A chatbots
- Information retrieval

Word embeddings: map each word to a point in high-dimensional (usually 100-300 dimensional) space. Captures semantic similarities, context, better generalized. Can build on top of Word2Vec vocabulary to be domain-specific through transfer learning.
Subword embeddings: break words into smaller units (subwords, character n-grams)
Document embeddings: Represent entire sentences, paragraphs,or documents as single vectors
Sentence vectorization: like normalization to use semantic "gravity" of sentence. Removes word order dimensionality.

- sentence_embedding = sum(embedding($word_i$))/n

Document embeddings

Word2Vec maps each word to vectors but can't represent whole documents. Embeds sentences, paragraphs, or documents as single vectors.
Distributed Memory (DM): Learns document vector by predicting target word from context words. Lower computation needed.
Distributed Bag of Words (DBOW): Learn document vectors by predicting words in document using only document ID.

Distributed Memory (DM)

Equivalent to CBOW approach where model predicts a target word by its surrounding words.

- good when word ordering matters. For longer structured texts
- Parameter dm = 1 (word order)

Distributed Bag of Words (DBOW)

Equivalent to skip-gram approach where model predicts surrounding words using a target word.

- good when word ordering does NOT matter. For shorter unstructured texts
- Parameter dm = 0 (no word order)

Macro average: Regular average
Weighted average: Multiply each point by its ratio within data.

Deep Learning

Intro to NLP - Fall 2025

Why deep learning for text?

Deep models are still main way to capture syntax, semantics, sarcasm, and multi-lingual nuance. Then inject small domain-specific data on transfer-learning models built on large corpora for higher accuracy

Prompt-based LLM: Comparable performance. No fine-tuning needed on many standard datasets.

Retrieval Augmented Generation (RAG) & prompt chaining: Push performance further without gradient updates.

When to use?

- high accuracy requirements - When 80% to 85% isn't sufficient
- Capture complex language - Sarcasm, context, long-range dependencies
- Domain adaptation - Fine tuning pre-trained models for specific domains
- Production scale - when training time investment will pay off

Modern alternatives:

- 0-shot LLM
- Few-shot learning
- prompt engineering

Preprocessing pipeline

Text preprocessing: 1st step before feeding data into deep learning model.

Keras Tokenizer: Creates word index from training texts. Training & test reviews converted to integers, representing word indices.

Padding: Sequences padded to fixed length (1,000 tokens always) to ensure the same shape for the model. This also helps GPU acceleration. Pre-padding vs post-padding may have their own advantages depending on architecture used.

1. Labels converted to categorical format for neural network output layer compatibility
2. All text data should be uniformly formatted for embedding & model input.

3. **Tokenization & Indexing**: Splits review into tokens (sequence of integer indices each representing a word in vocabulary) using Keras tokenizer. num_words = parameter for getting top n most frequent words across dataset
4. **Out-of-vocabulary (OOV) words**: Any word outside of num_words is replaced with special token. Ensures robust handling of rare/unseen words. This reduces model complexity & focuses on learning most relevant vocabulary

CNN

1-D convolutional layers: Sliding window/n-gram detectors to identify local patterns in text

Max pooling layers: Choose most salient feature from each filter that reduces dimensionality & focus on key signals.

CNNs: Efficient for text classification with fast training & inference times. Effective when local word patterns can strongly indicate class labels.

Architecture: Balances depth with efficiency. Good for large-scale text dataset.

- 3 Conv1D layers
- 3 MaxPooling1D layer
- 1 GlobalMaxPooling1D layer
- 1 Dense layer
- Could add dropout & batch normalization for regularization & better generalization.

Loss: Could be above 1. Lower is better

Accuracy: 0-1. Higher is better but only a good metric if dataset is evenly balanced across all possible labels.

Insights:

- Pretrained embeddings accelerate early learning & improve on small datasets
- Task-specific embeddings (trainable) may overfit less on validation but needs more data for best performance.
- Pre-trained vs trainable embeddings depends on data volume, domain similarity, and computational resources. Experiment with both
- Hyperparameter tuning - Change filter size, dropout rate. Could impact model effectiveness

Usage - Fast inference. Excels at capturing local patterns & n-gram features, efficient for large datasets.

LSTM

Long Short Term Memory (LSTM): Input, forget, and output gates to mitigate vanishing gradient problem & learn long-term dependencies. Preserves order information. Hidden

state in RNNs carries information across tokens, allows for model to remember previous words.

Architecture

- Embedding layer
- LSTM layer
- Dense output layer with sigmoid activation (for binary classification)
- Dropout & recurrent dropout (0.2 each) applied to prevent overfitting & towards generalization.

Can use early stopping if validation accuracy stops improving.

Usage - Models long-range dependencies & sequential context. Ideal for tasks where word order matters

Hybrid (Conv + Bi-LSTM)

Combines both architectures' strengths.

Usage - In research for state-of-the-art results.

Transformers for NLP

Transformers: Introduced in 2017 for processing sequential data like language. Uses self-attention to process all parts of input sequence at once, not step-by-step. Transformer can visualize all words and their relation to every other word no matter their position.

Ex: BERT (understands), GPT (generator)

Transformers' core parts

- **Self-attention layer**: Determines how much attention each word in sentence should pay to every other word. Model can understand their relationships within context
- **Multi-head attention**: Runs multiple self-attention operations in parallel. Lets model capture different relationship types (Ex: geographic, action, description) simultaneously. This helps transformer understand complex text patterns
- **Feed-forward networks**: After attention step, each word's context-aware representation passed through small neural network to further refine info
- **positional encoding**: Add special signals to each word to indicate its sentence position ensure model understands word order & sequence

Advantage:

- *Global context* - Every word's relationship/dependency with every other word is captured
- *Parallel processing* - All positions in sequence are processed at once so training is faster and more scalable than sequential models
- *Transfer learning* - Can be pre-trained on massive text data then fine-tuned for specific tasks with minimal extra data.

Transformer steps:

1. Input = "The movie was good!"
2. Tokenization = ["The", "movie", "was", "good!"] tokens
3. Embedding & positional encoding = Convert each token into vector, encode it's index position too.
4. Self-attention = Compute how much each word should pay attention to the other words
5. Stacked layers = Many layers of attention & feed-forward networks refine understanding
6. Output = Contextualized representation of sentence for classification/translation use

BERT

Pipeline (automated pipeline on TensorFlow Hub):

1. Tokenization - Split into WordPiece tokens
    a. Special tokens - Adds [CLS] at the very start & [SEP] at very end of every sequence so that BERT can understand start & stop.
2. Padding/Truncation - Truncate/pad input to common length (usually 128 for efficiency/compatibility)
3. Output Tensors: The preprocessing model outputs three key tensors for each input:
    a. input_word_ids: Integer IDs representing each token (including special tokens and padding).
    b. input_mask: Binary mask indicating which positions are actual tokens (1) vs. padding (0).
    c. input_type_ids: Segment IDs (for distinguishing between multiple sentences; all zeros for single-sentence tasks like sentiment analysis).

Def: Vocab
Def: Vocab
Def: Vocab

Large Language Models

Intro to NLP - Fall 2025

Feature	Word2Vec / Doc2Vec	BERT (Contextual)
Vector per word	Single (static)	Varies by context
Handles ambiguity	No	Yes
Sentence/document	Doc2Vec for documents	Yes, with sentence embeddings
Training paradigm	Predicts context/words	Masked language modeling
Use cases	Retrieval, clustering	QA, classification, search
Model architecture	Shallow neural networks	Deep transformers

AI Language Models

1. 2012 - Deep neural networks (CNNs) wins ImageNet, starting AI boom
2. 2017 - Transformer architecture introduced
3. 2019 - GPT2 released and generates human-like articles
4. 2020 - GPT3 launches with 175 parameters & text generation
5. 2022 - ChatGPT released
6. 2023-2025 - Improved models, Claude, DeepSeek, Gemini, Llama4, Grok

Conversational AI is now available and adoptable anywhere.

- AI for translation, summarization, and content creation
- **Multimodal AI**: Input natural language generate code, text, image, different medias, etc. Or a combo of everything.

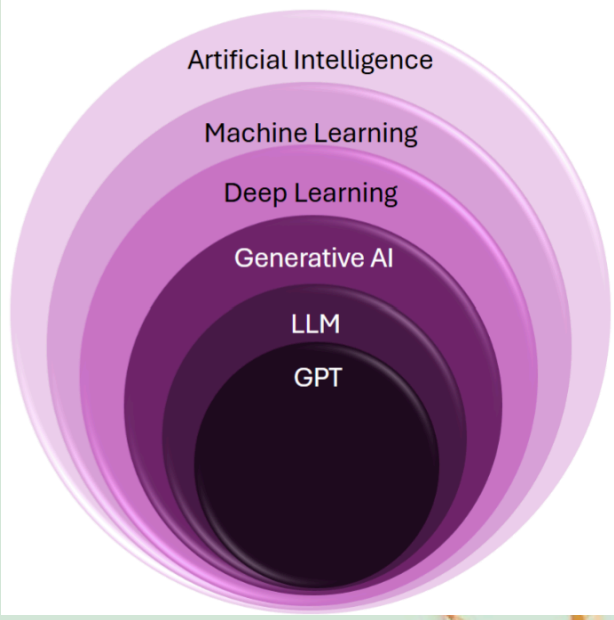
Language AI: Special branch of AI that focuses on letting computers understand, process, and generate human language in both written/speech forms.

history

Early approaches - Simple rule-based systems & keyword matching which couldn't capture real-world language

Breakthrough solution - Language represented in structured machine readable formats like vectors & embeddings, letting algorithms analyze & generate text more effectively

Large Language Models (LLMs): AI models trained on massive datasets to generate & understand human-like language.



RNNs with Attention

Recurrent Neural Networks (RNNs): Early neural models for processing sequences, capturing context in text 1 token at a time.

- Limitation - Can't do long-range dependencies, difficult to train in parallel
- **Attention mechanism**: Lets model focus on relevant parts of input sequence to improve translation & sequential tasks.
- **Attention**: Lets model selectively weight importance of different words, lead to more accurate & flexible understanding
- **self-attention**: Model needs to determine what "it" refers to in a sentence. Self-attention lets every word in sentence "look at" every other word. "it" attends most strongly to another word in sentence "animal", so model understands this reference. Distance of words to each other in sentence doesn't matter.

encoder-only model (BERT)

Goal - Classify sentiment in a movie review.

1. BERT tokenizes input movie review.
2. Pass through multiple encoder blocks
3. CLS token at input's beginning accumulates info from whole sentence
    a. **Classification token (CLS)**: Token introduced in BERT (encoder-only model from a Transformer framework)
4. Final CLS token's state helps predict sentiment

Benefit - BERT's encoder-only design best for extracting contextual features for classification/search

decoder-only model (GPT)

Goal - Create something new

1. Takes prompt, generate next word, one at a time
2. At each step, use self-attention to consider all previous words in sentence
3. repeat steps

Benefits - Generates new text, answers question, complete code

Examples

Encoder-decoder model for translation

1. **Encoder**: Encodes for example, English using self-attention to create rich contextual representation
2. **Decoder**: Generate for example, Chinese translation word by word, attending to previously generated words & encoder's output.

Benefit - Ideal for sequence-to-sequence tasks like translations and summarizations

BERT masked language model

1. Model receives text. Replaces one word/token with "[MASK]"
2. Uses self-attention, considering entire context to predict most likely original word/token before [MASK]

Benefit - Pretraining strategy lets BERT understand language context.

Generative LLM Chatbot

1. Feed user prompt into decoder-only LLM
2. Model generates complete, contextually appropriate output

Benefit - Generate human-like reponses & assist in real-world communications

Reasoning model

1. Has a deep reasoning model
2. This mode is for harder problems, not for typical queries

Benefits - solves more complex problems.

Multimodel LLM

1. Processes another media ex: image
2. Image recognition
3. Text-based decoder generates something ex: image description, based on image

Benefits - Modern LLM handles text & images
Vision, audio, code, video

parameters

Context: Prompt + Attached files. Counted using tokens
Parameter: Weights, inputs, and outputs, biases

Model	Developer	Parameters	Context	Key Strengths	Release
GPT-5	OpenAI	Unknown	Unknown	General reasoning, safety	Aug 2025
Grok 4	xAI	Unknown	256K	Real-time web access	July 2025
Llama 4 Scout	Meta	17B active	10M	Ultra-long context	April 2025
Claude 4 Opus	Anthropic	Unknown	200K	Coding excellence	May 2025
Gemini 2.5 Pro	Google	Unknown	1M	Multimodal reasoning	June 2025
Qwen 3	Alibaba	235B	262K	Multilingual, open	April 2025
DeepSeek R1	DeepSeek	37B active	128K	Reasoning, cost-effective	Jan 2025

Mixture of Experts (MoE): Only activates subset of parameters
Extended context: Standardizing 1M+ token windows
Multimodal by default: Text + image + video + audio
Agent capabilities: Built-in tool use & planning

Quantization

Quantization: Reduce model size by lowering numerical precision of parameters/activations. Has various time for quantization post-training, quantization-aware training, static vs dynamic quantization, mixed precision, etc. Ex: From 32-bit floating point to 8-bit or even 4-bit ints.

Benefits:

- Smaller storage
- reduce memory usage/bandwidth
- faster inference
- can be deployed on less powerful hardwares

Limitations:

- Slight loss in accuracy
- Naive quantization can degrade performance badly if calibration/training adjustments not made

Ex: OPT-350M, Mistral-7B-Instruct, PrunaAI

context window

context window: MAx tokens LLM can process at once. Ex: 4000-token window handles 3000~4000 words. Longer windows allows for more prompt & context space.

model size & hardware requirements

- RAM/VRAM - Model size = parameter count
- Small models = 1-7 Billion parameters on regular GPU/CPU computers
- Large models = 70+ Billion parameters needs cloud/high-performance computers

open source LLMs

- Meta's Llama 3.¼
- Microsoft's Phi-¾
- Mistral Large 2
- StableLM
- Tulu 3
- Falcon 180B
- BLOOM
- Ollama

Model Name	Parameter Size	VRAM (Quantized)	Key Strengths
Llama 3.1 8B (Quantized)*	8B	~7–10 GB	Chat, summarization, reasoning
Gemma 7B (Quantized)	7B	~6–8 GB	Text generation, multilingual
Phi-3 Mini (Quantized)	3.8B	~7–8 GB	Reasoning, coding, efficient
Qwen2.5 7B	7B	~6 GB	General, multilingual
Mistral Small 22B (Quant.)	22B	~15 GB	Creative writing, fast queries
Falcon 7B	7B	~8–10 GB	Reasoning, research
StableLM 7B	7B	~8 GB	Prototyping, general purpose
TinyLlama 1.1B	1.1B	~4–5 GB	Low-resource, fast testing

-

- LM-studio
- HuggingFace Model Hub

huggingFace

General code:

```
from transformers import AutoModelForCasualLM, AutoTokenizer
model_id = “model_id”
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(model_id)
```

Loading model:

```
from transformers import AutoModelForCasualLM, AutoTokenizer
model_id = “model_id”
model = AutoModelForCasualLM.from_pretrained(model_id,
device_map=“cuda”, torch_dtype=“auto”,
trust_remote_code=False)
```

-

```
tokenizer = AutoTokenizer.from_pretrained(model_id)
```

Loading model:

```
from transformers import pipeline
generator = pipeline(“text-generation”,model=model,
tokenize=tokenizer, return_full_text=False,
max_new_tokens=500, do_sample=False)
```

LLM tokenization

LLM Tokenization: LLMs breaks words down to subwords (prefixes, suffixes, or character groups) or character-level tokenization.
Ex: “vocabulary” split into “vocab#####” and “#####bulary”
Goal - Reduces vocabulary size, handles new & rare words. Accommodates string, misspellings, code, emojis, etc. Reduces Out Of Vocabulary tokenization

1. Enter natural language prompt into window
2. LLM tokenizes it into sequence of token IDs
3. LLM will only ever see & process token IDs.
4. LLM outputs response

Ex: Model name is “model-4K-instruct” which will mean it has 4K token window and tokens that exceed 4K will be truncated.

Role assignment: Add "<|assistant|>" to whatever natural language prompt you input into LLM. This helps LLM understand its job.

Compare tokenizers

Compare how LLMs handle emojis, code, math, different languages, and or special characters, compare efficiency, context window usage.
Ex: text = "*University of Missouri and CAMPUS LIFE 🎓 Mizzou Tigers show_tokens False None elif == >= else: two tabs:" " Three tabs: " " 12.0*50=600*"

WordPiece tokenization

BERT-uncased tokenization = `[CLS] university of missouri and campus life [UNK] mi ##zzo ##u tigers show _ token ##s false none eli ##f = = > = else : two tab ##s : " " three tab ##s : " " 12 . 0 * 50 = 600 [SEP]`

- *Special tokens* - Adds [CLS] (classifier token) & [SEP] (separator token)
- *Emoji handling* - Uses [UNK] to represent 🎓
- *Subword splitting* - "Mizzou" = `mi ##zzo ##u`
- *Case normalization* - "Mizzou" = `mi ##zzo ##u`
- *Math expression* - Each operator & number gets separate tokens

Strength - Good for classification, handling unknown words by subwords
Weakness - Poor emoji support, lose capitalization

Byte-Pair Encoding

GPT2 tokenization = `University of Missouri and C AMP US LIFE ▯ ▯ ▯ ▯ M izz ou Tigers show _ t ok ens False None el if == >= else : two tabs :" " Three tabs : " " 12 . 0 * 50 = 600`

- *Special tokens* - No sentence markers. Directly process text
- *Emoji handling* - Processes 🎓 as distinct tokens (multi-byte)
- *Subword splitting* - "Mizzou" = `M izz ou`
- *Case normalization* - Maintains capitalization. "Mizzou" = `M izz ou`
- *Math expression* - Each operator & number gets separate tokens
- *Spacing sensitivity* - Handles tabs & spaces as separate tokens

Strength - Good unicode/emoji support. Preserve formatting.
Weakness - More tokens for some words, sensitive to spacing

SentencePiece Encoding

T5/Flan-T5 tokenization = `University of Missouri and C AMP US LI FE <unk> Mi zzo u Tiger s show _ to ken s Fal s e None e l if = = > = else : two tab s : " " Three tab s : " " 12. 0 * 50 = 600 </s>`

- *Special tokens* - More efficient splitting (fewer total tokens). `</s>` at end of sequence.
- *Emoji handling* - `<unk>` for unsupported emojis like 🎓

- *Subword splitting* - “Mizzou” = `Mi zzo u`
- *Case normalization* - Maintains capitalization. “Mizzou” = `Mi zzo u`
- *Math expression* - Each operator & number gets separate tokens
- *Spacing sensitivity* - “`12.`” grouped together before splitting

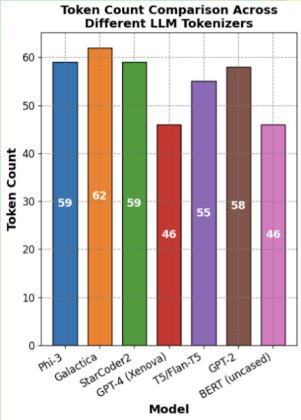
Strength - Efficient token usage. Good for multilingual applications
Weakness - Some emoji limitations. Aggressive subword splits

Advanced Byte-Pair Encoding

GPT4 tokenization = `University of Missouri and C AMP US LIFE`
`🎓 M izz ou Tigers`
`show _tokens False None elif == >= else : two tabs :" " Three tabs :" "`
`12 . 0 * 50 = 600`

- *Special tokens* - No sentence markers. Directly process text
- *Emoji handling* - Processes 🎓 as distinct tokens (multi-byte)
- *Subword splitting* - improved efficiency “elif” = `elif`
- *Case normalization* - Maintains capitalization. “Mizzou” = `M izz ou`
- *Math expression* - Similar granular splitting into math expressions
- *Spacing sensitivity* - Preserves tab spacing

Strength - Strong code/unicode/emoji support. Efficient & accurate.
Weakness - More tokens needed for some text



Model000	Token Count	Efficiency
T5/Flan-T5	46	Most efficient (fewest tokens)
BERT (uncased)	46	Most efficient (fewest tokens)
StarCoder2	55	Balanced
GPT-2	58	Balanced
Phi-3	59	Balanced
GPT-4 (Xenova)	59	Balanced
Galactica	62	Least efficient (most tokens)

Aspect	BERT WordPiece	GPT-2 BPE	T5 SentencePiece	Modern Models (Phi-3, GPT-4, etc.)
Emoji Support	❌ [UNK]	✅ Multi-byte	⚠️ <unk>	✅ Multi-byte
Special Tokens	[CLS], [SEP]	None	</s>	Minimal
Case Sensitivity	❌ Lowercased	✅ Preserved	✅ Preserved	✅ Preserved
Code Handling	Basic	Good	Good	Excellent (StarCoder2, etc.)
Math Expressions	Character	Character	Mixed	Character
Token Efficiency	High	Balanced	High	Balanced (except Galactica: Low)

WordPiece & SentencePiece are most efficient.

Galactica least efficient (more tokens = more computation)

contextualized word embeddings

Contextualized embeddings: Represent each token dynamically based on specific sentence context. Understands language ambiguity & nuanced word meanings. Not like static word vectors that always gives same representation regardless of context.

- **static embeddings (word2vec, GloVe)**: "bank" always has same vector
- **contextualized embeddings (BERT, GPT)**: "bank" has different vectors

Shape: [# of documents, # of tokens, # of dimensions], Ex: [1,4,384]

- **dimensions/model hidden size**: Length of numeric vector. More dimensions captures richer and more precise meaning. Similarities aren't lost. Smaller dimensions = fast computation. Large dimensions = slow computation.

Model / Embedding	Embedding Dimension	Description/Typical Use
word2vec / GloVe	300	Static word embeddings
InferSent / Doc2Vec	512	Sentence/document embeddings
BERT-base / Sentence Transformers	768	Token, sentence, para. embed.
OpenAI text-embedding-3-large	3072	Universal text embedding API
GPT-2 (XL) / GPT-3 (Davinci)	1600–12288	LLM token embeddings

from token to sentence

Sentence-level or document-level representations for richer semantic understanding.

Def: Vocab

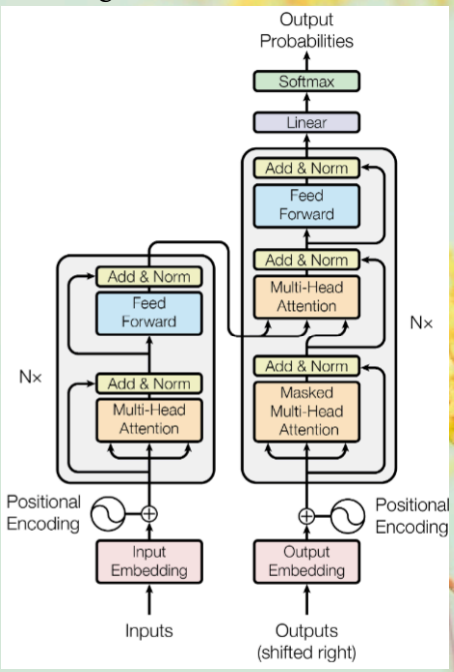
Def: Vocab

Inside LLMs

Intro to NLP - Fall 2025

Introduction

LLMs: Built on Transformer architecture. Models process text by breaking text down into tokens and turning tokens to embeddings (mathematical representations) to capture meaning & context.

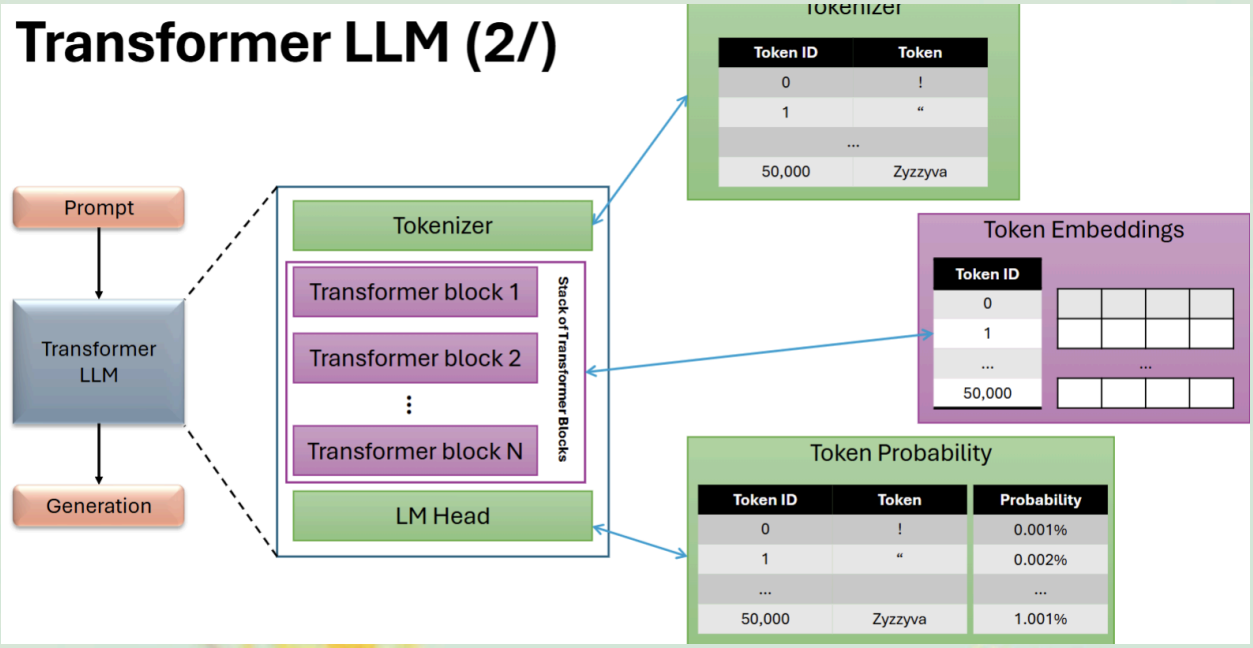


Transformers: Operate using layers to let each word attend to every other word (self-attention), capturing relationships & context.

<u>3 main components of Transformer LLM</u>:

1. **Tokenizer** - Breaks text into token IDs using vocabulary
2. **Stack of Transformer Blocks** - Performs core processing
3. **Language Modeling Head (LM Head)** - Convert output to token probabilities. Other "head" variants could be sequence classification heads & token classification heads for different tasks.

Each token generation requires 1 complete forward pass through these components



Transformer benefits:

- parallel token processing - Each token has own computation stream
- context length limits - Model can only process fixed # of tokens (Ex: 4K)
- attention interactions - Tokens can attend to previous positions
- Important insight - Only last token's output vector used for next token prediction, but all previous computations needed for attention mechanism

Input & output of trained transformer LLM

Ex: Phi-3 Model Structure...

```
Phi3ForCausalLM(
(model): Phi3Model(
        (embed_tokens): Embedding(32064, 3072, padding_idx=32000)
        (layers): ModuleList(
                (0-31): 32 x Phi3DecoderLayer(
                (self_attn): Phi3Attention(...)
                (mlp): Phi3MLP(...)
        )
)
(norm): Phi3RMSNorm((3072,), eps=1e-05)
)
(lm_head): Linear(in_features=3072, out_features=32064, bias=False)
)
```

Summary

- Model contains 32,064 tokens in its vocabulary
- Each token represented by 3,072 dimensional embeddings.
- Pass text through 32 transformer layers
- RMSNorm: Approx 15% faster than LayerNormm with identical performance
- No bias in lm_head - Modern optimization to save parameters
- Pass final language modeling head to produce output

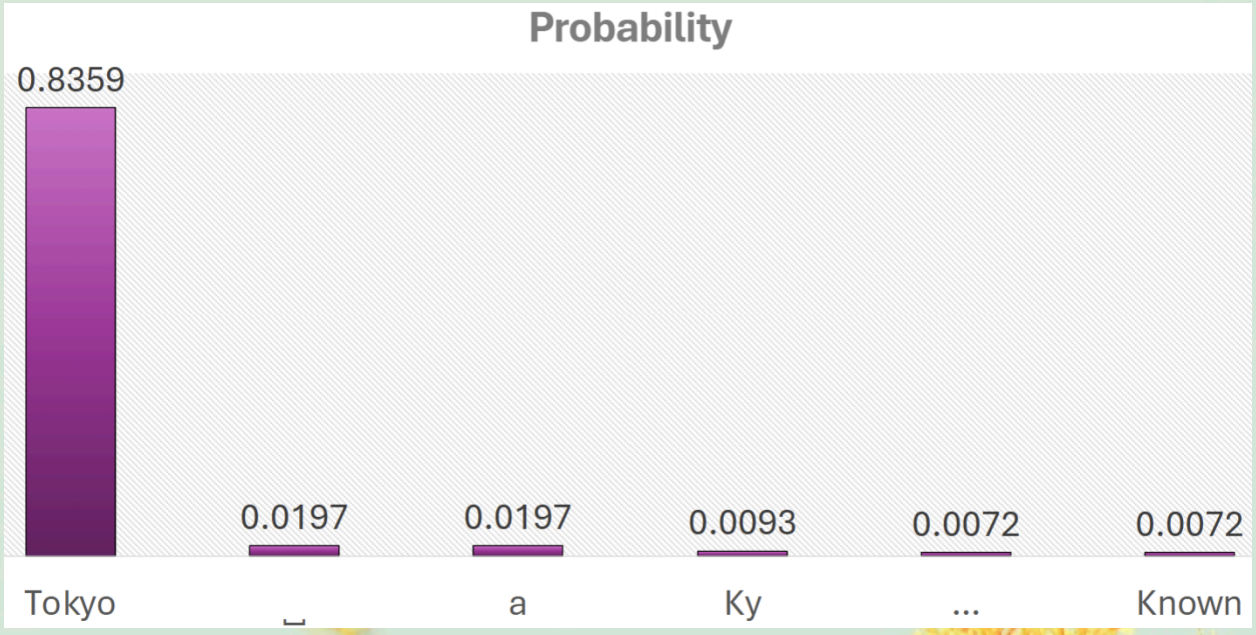
LLM Pipeline

Key Tensor Transformations:

- **Text:** *"The capital of Japan is"* (human readable)
- **Token IDs:** [450, 7483, 310, 5546, 338] (discrete integers)
- **Embeddings:** [1, 5, 3072] (dense numerical vectors)
- **Hidden States:** [1, 5, 3072] (contextually enriched)
- **Logits:** [1, 5, 32064] (probability scores for all vocab)
- **Prediction:** *"Tokyo"* (82.8% probability)

1. Text tokenization & Embedding - Subword tokenization
2. Each token embedding passed through self-attention layers to add on contextual info from other positions
3. Each token passed through feedforward networks to apply learned transformations to capture complex pattern
4. Each token passed through normalization layer & residual connections to maintain training stability
5. Language Modeling Head & Probability distribution - Final transformer layer outputs passed to LM head that applies linear transformation to map 3,072-dimensional vectors to probability scores across 32,064-token vocabulary
    a. **logits**: Raw unnormalized score produced by model's final layer for each possible token. Score reflects how likely each token is before applying softmax. Format for HuggingFace. Output after passing through linear layer.
6. Token selection & decoding - Model select next token from probability distribution using decoding strategy.
    a. **Greedy decoding**: Temperature = 0, always choose highest probability token. So very deterministic outputs.
        i. **temperature**: Introduces randomness by modifying softmax distribution

b. **Top-k/Top-p sampling**: Restricts sampling to most likely tokens for better quality



Selecting final output token

Decoding/sampling: Selection process for final model output. For multi-token output, each new output is appended to input and fed into model again to predict the next token.
Considerations:

- Pick highest probability token? (should you have deterministic or variant output?)
- How to balance predictability with creativity?
- Tradeoff between quality & diversity.

temperature parameter

Temperature controls sharpness of probability distribution. Probabilities divided by temperature before applying softmax.

$$P(token) = \frac{e^{logit/T}}{\sum e^{logit/T}}$$

- temperature = 0. Equivalent to greedy decoding
- temperature <1. More focused, predictable outputs
- temperature =1. Original probability distribution
- temperature >1. More random & creative outputs.

greedy decoding

Deterministic - Fastest, predictable. Same input returns same output every time. This uses Greedy Decoding with temperature=0.

Limits:

- *repetition & loops* - Get stuck in repetitive patterns. Ex: "Cat sat on the cat sat on the..."
- *generic output* - uses most common phrases from training data. lacks creativity/diversity
- *suboptimal global solution* - Local optimal choice may lead to globally poor choice. Can't "undo" previous suboptimal weight/parameters
- *missing context dependencies* - May ignore subtle contextual clues that need less probable tokens.

sampling-based decoding

Always pick highest probability token but sample from probability distribution. Sampling allows bigger pool to select from. Outputs are more human-like.

Ex: "Dear" has 40% probability so pick "Dear" 40% of the time.

Control Candidate pool

Top-k sampling: Only consider top k most probable tokens. Ex: Top-50 sampling only select from 50 most likely tokens.

- pro - prevent selection of very unlikely tokens
- limit - fixed k is too big when model is confident & too small when model is uncertain

Top-p sampling: Sample from smallest set of tokens whose cumulative probability >=p.

- pro - dynamically adjustment based on model confidence

Ex: p=0.9, may include 3 tokens when model is confident, 20 when uncertain

Strategy	Speed	Determinism	Creativity	Quality
Greedy	Fastest	100%	Low	High for factual tasks
Low Temp	Fast	High	Low	High coherence
High Temp	Fast	Low	High	Variable quality
Top-k/Top-p	Moderate	Low	Moderate	Balanced

<u>Parameter selection common practices</u>

- temperature - Start with 0.7-1.0 for balanced output
- top-p - 0.9 is common default

- Top-k - 40-50 typical range

Computational costs

- Greedy - Minimal overhead
- sampling - Needs random # generation
- advanced methods - Additional probability calculations

Consistency vs Creativity

- Use seed for reproducible sampling
- Consider application requirements

beyond single-token decisions

Autoregressive nature: Each token selection affects ALL future choices. Wrong choice early can derails entire response.
Context building: Selected tokens become part of context for next prediction. Sampling strategy shape flow of conversation.
User experience: Same prompt can produce different responses.

KV-Cache performance demonstration

When generating each new token, recomputing all previous states is inefficient for long outputs, taking $O(n^2)$ time for every generation.
KV (key-value) Cache solution: Stores keys & values for all previously processed tokens from attention mechanism. Speedup = O(n). On each new step, only computations for latest new tokens needed.

- Turn on feature by doing use_cache=True

Configuration	Mean Time	Std Dev	Throughput	Statistical Confidence
With KV cache (use_cache=True)	4.73s	±0.273s	21.1 tokens/sec	7 runs, highly reproducible
Without KV cache (use_cache=False)	35.2s	±0.144s	2.8 tokens/sec	7 runs, consistent results
Performance gain	7.4× faster	-	644% improvement	Statistically significant

Save common prompt answers in cache so they can be more quickly fetched.

Text classification with LLMs

Classification types:

- Single-label (classify 1 label) VS Multi-label (classify many labels)
- Flat taxonomies VS hierarchical trees/DAGs (positive-> very positive vs little positive)
- Binary vs multiclass vs extreme multi-label (>=10K labels)

Benefits using LLMs:

- ☐ LLMs allow for 0-shot or few-shot transfer
- ☐ Dense contextual embeddings capture semantics better than handcrafted features
- ☐ Generative decoding offers natural-language rationale, improves explainability

Task Type	Example Dataset	Recommended LLM Strategy	Key Metric
Sentiment Analysis	IMDb, Amazon Reviews	Fine-tuned BERT or GPT-4 few-shot	Macro F1
Intent Detection	CLINC-OOS	Frozen embeddings + logistic head	Accuracy
Toxicity Detection	RealToxicityPrompts	Generative CoT + verifier	AUROC
Language Identification	WiLI-2018	Zero-shot GPT-4	Accuracy

New specialized tasks with LLMs

Hierarchical categorization: TELEClass expands upon taxonomies with LLM-generated features for minimal supervision

Extreme Multi-label: AmazonCat-13K solves via Label Attention & Correlation Networks (LACN)

Long-form sparse signal: Domain-specific reports handled by hierarchical term-pooling models.

failure-mode coding in maintenance logs: GPT3.5 finetune outperforms legacy models by 20 pp F1

data preparation

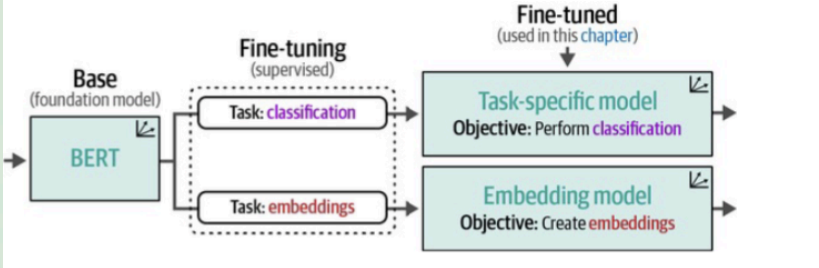
Challenge	Classic Fix	LLM-Era Fix	Caveat
Label scarcity	**Crowdsourcing:** employ large pools of human annotators to label data, often via online platforms.	**GPT-3.5/4 synthetic labels:** prompt a powerful LLM to pseudo-label massive unlabeled datasets in a fraction of the time.	**Quality variance:** LLM-labeled data may have uneven quality across classes, especially for subjective or nuanced tasks.
Privacy	**Redaction:** manually or automatically remove personally identifiable information (PII) from datasets before release or use.	**DP-guided generation:** use differentially private text generation frameworks to synthesize data that preserves privacy guarantees while retaining overall data utility.	**Compute overhead:** implementing and monitoring true DP approaches increases computational cost, and utility may drop when strict privacy is enforced.
Annotation cost	**Passive tagging:** sample and sequentially send unlabeled examples to annotators, labeling all with equal priority.	**Uncertainty-based active learning (AL):** models query annotators for only the most ambiguous, informative, or uncertain samples, guided by prediction uncertainty, drastically reducing total human labeling effort.	**Needs calibrated confidence:** AL effectiveness depends on accurate model confidence scores; poorly calibrated models can yield suboptimal sample selection.

Main strategies for LLM classification

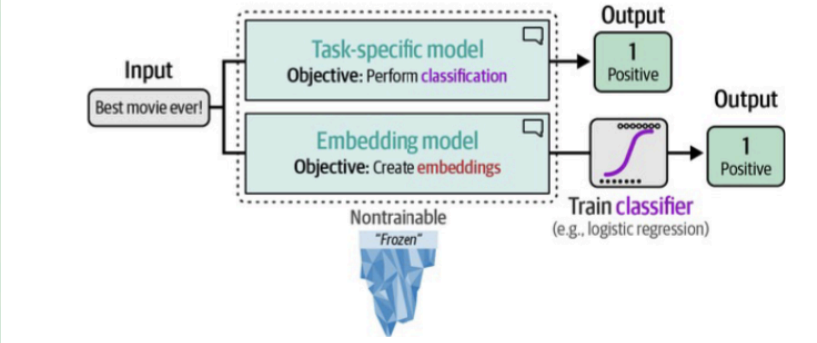
Task-specific models: Fine-tuned directly on labeled data

General-purpose models: Embeddings models used by light-weight classifiers. All experiments

- Freeze transformer weights to emphasize inference time reuse over retraining.



A foundation model can be fine-tuned for tasks such as classification or embedding generation.
Image Source: Alammar J, Grootendorst M. *Hands-on Large Language Models: Language Understanding and Generation*. First edition. O'Reilly Media, Inc; 2024.



Use a task-specific model for direct classification, or leverage general-purpose embeddings for indirect classification.
Image Source: Alammar J, Grootendorst M. *Hands-on Large Language Models: Language Understanding and Generation*. First edition. O'Reilly Media, Inc; 2024.

representation model landscape

Encoder-only transformers - better at classification bc of bidirectional context capturing & compact size

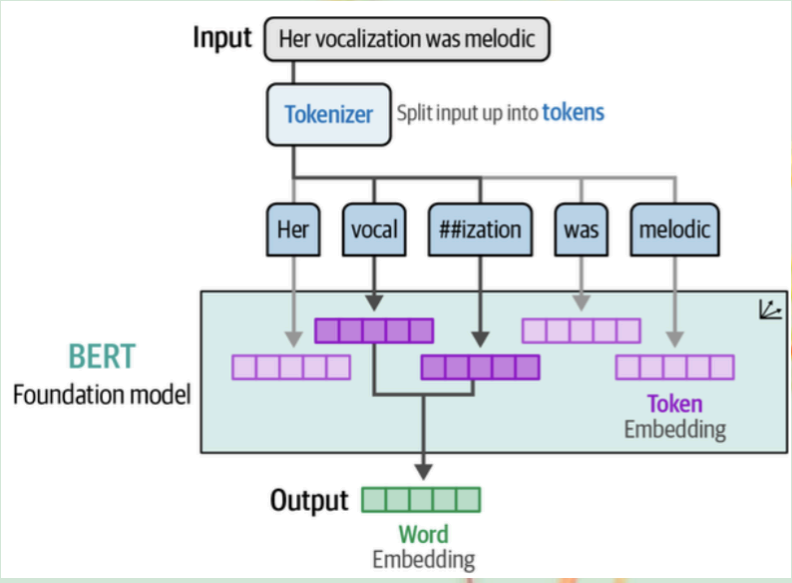
Decoder-only transformers - Better at open-ended tasks but larger & slower for pure classification.

New BERT variants try to improve parameters, speed, and accuracy while maintaining high representation quality.

Model	Release Date	Parameters	Organization	Key Innovation
BERT	October 2018	110M/340M	Google	First bidirectional transformer encoder
RoBERTa	July 2019	125M/355M	Meta/Facebook	Optimized BERT training (removed NSP, dynamic masking)
ALBERT	September 2019	12M-235M variants	Google	Parameter sharing + factorized embeddings
DistilBERT	October 2019	66M	Hugging Face	Knowledge distillation (40% smaller, 60% faster)
DeBERTa	June 2020	134M-750M	Microsoft	Disentangled attention mechanism

Model	Params	Pros	Cons
BERT-base-uncased	110 M	Versatile, well-studied	Moderate latency
RoBERTa-base	110 M	Better pre-training corpus	Larger memory than distillations
DistilBERT-base-uncased	66 M	60% faster, smaller	Slight F1 drop vs RoBERTa
DeBERTa-base	140 M	High accuracy	Slower inference
bert-tiny	14 M	Edge devices	Lower ceiling
ALBERT-base-v2	12 M	Minimal memory	Needs longer fine-tunes for peak F1

Category	Questions to Ask	Why It Matters
Language & Domain	Is the pre-training data similar to Rotten Tomatoes prose?	Domain mismatch hurts F1.
Architecture	Encoder-only vs encoder-decoder vs decoder-only	Impacts speed & memory.
Size	How many parameters?	Affects latency and cost.
Licensing	Can we deploy commercially?	Compliance risk.
Hardware Budget	GPU vs CPU, edge vs cloud	Determines viable model footprint.



Metric	Formula	Interpretation
Precision (P)	TP / (TP + FP)	Share of predicted Fresh that really are Fresh.
Recall (R)	TP / (TP + FN)	Share of actual Fresh captured.
Accuracy	(TP + TN)/Total	Overall correctness percentage.
F1 Score	2·P·R / (P + R)	Harmonic mean balancing P & R.

Class	Precision	Recall	F1	Support
Rotten	0.76	0.88	0.81	553
Fresh	0.86	0.72	0.78	553
Weighted Avg	—	—	0.80	1066

Text classification with embeddings

All-mpnet-base-v2 embedding model: 768-dimensional vector. Fast, memory and accuracy across tasks. Good for when 1 encoder must manage multiple services: search, clustering, tagging.

Sequence-to-Sequence

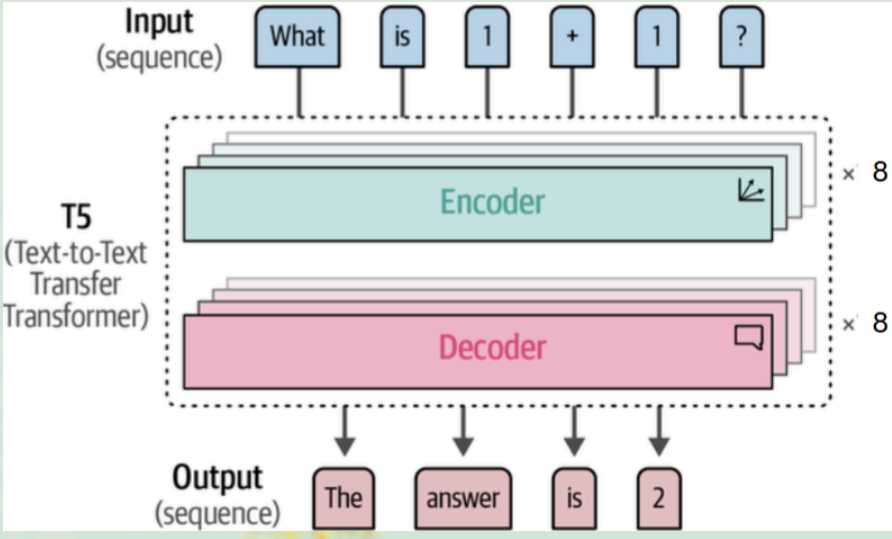
Classification treated as translation = review -> label string
Input - token sequence represent raw review
Output - Token sequence = label text
Makes label set open-ended so that model can generate “neutral”, “mixed”, or new categories if asked.

prompt engineering

- ☐ Have concise & clear instruction and prompts
- ☐ Have strict format rules for output (makes it easier to parse)
- ☐ Use few-shot examples (important to separate instructions from the examples and examples from actual input)
- ☐ Assign a role in system message
- ☐ Separate instructions from actual input prompt/data
- ☐ **Iteration cadence**: Run tests on various prompt. Log error clusters for data driven refinement.

Text-to-text

T5 Text-to-text transfer transformer: Uses Transformer architecture. Structured similarly to original Transformer model. Has stack of 8 encoders & 8 decoders.



Performance metrics & evaluation

Standard metrics: Use classification metrics like accuracy, precision, recall, and F1-score.
benchmark: Compare performance of different models on specific dataset to find most suitable model for needs
Data specificity: Different domain datasets may have various success on models. Finetuning or using RAG may be needed for specialized domains.
Iterative evaluation: Regularly evaluate performance of chosen model & prompt as new models are released.
Human evaluation: Good for subjective tasks like sentiment analysis.

Prompt engineering

Intro to NLP - Fall 2025

Chain prompts

Allows for more customization at each step (set different temperatures for each task), isolate error, and composable systems where many naming models can be parallelized.
Response validation: 2nd LLM critiques 1st output. Can be fact-checking
Parallel prompts: Many LLMs ran concurrently & merged later.
Chain of Thought (CoT): Instruct LLM to do step-by-step reasoning
Tree-of-Thought: Branch and evaluate strategy for complex problems

Practical workflow

1. Draft minimal workflow
2. Identify missing components, adding iteratively
3. Give few-shot examples if outputs deviate
4. Split tasks across chained prompts
5. Validate with another LLM or rule-based checks
6. Automate workflow with LangChain or LlamaIndex

Reasoning with generative models

LLMs are good at pattern matching & text generation but struggle with complex reasoning tasks that needs multi-step logical thinking, mathematical problem solving, causal analysis and inference, and strategic planning and decision making.
A good workflow - Give model a problem, provide it the answer (with step by step solving techniques). Give model a new problem.
Why it works?

- ☐ *Computational distribution* - Spreads thinking across multiple tokens
- ☐ *Pattern recognition* - Show how model approaches similar problems
- ☐ *Error Reduction* - step-by-step process catches logical mistakes
- ☐ *Transparency* - Make reasoning process visible & debuggable

Benefits:

- ☐ *Improved accuracy* - 10-30% better performance on reasoning tasks
- ☐ *Stable outputs* - More consistent results across many runs
- ☐ *Better explanations* - Clear reasoning path for debugging

Disadvantages:

- ☐ Could be slower
- ☐ Higher computational costs
- ☐ Higher # of output tokens needed.

zero-shot chain of prompt

Emphasize in user instructions "Think step-by-step." Don't give any examples of chain of thought reasoning steps.

self-consistency method

Outputs vary due to randomness. Sample multiple generated outputs for better results and use majority voting. This can improve accuracy but requires more computational resources and time.

tree-of-thought

Break down complex problems into intermediate steps. At each step, generate multiple answers. Vote for correct answer. Continue to next step. More computationally expensive & slow.

- Can use different models to generate different answers at each step

Method	Complexity	Speed	Use Case
Chain-of-Thought	Medium	Fast	Mathematical problems, step-by-step reasoning
Zero-Shot CoT	Low	Fast	When no examples available
Self-Consistency	Medium	Slow	Higher accuracy requirements
Tree-of-Thought	High	Very Slow	Complex creative tasks

output verification & control

In production, outputs must be reliable and controlled. This prevents failures and maintains quality of system.

Generative models usually generate inconsistent outputs.

Grammar

Grammar: Template to guide output format of generative AI. Generate only outputs for the placeholder texts. No guarantees but can make it more robust.

Packages:

- Guidance
- Guardrails
- LMQL - Language Model Query Language

Aspect	Examples (Few-shot)	Grammar Constraints
Control Level	Guidance only	Strict enforcement
Reliability	Model-dependent	Guaranteed
Implementation	Simple	Complex
Flexibility	High	Limited
Performance	Fast	Slower

Advanced Text Generation

Intro to NLP - Fall 2025

introduction

LLM performance can be improved without fine-tuning by using techniques like prompt engineering and system design.

Progress since LLMs:

- Model I/O - More efficient strategies like quantized model to load & run LLMs, including on consumer hardware
- Memory - Lets LLMs remember context across interactions
- Agents - LLM-powered "workers" can take actions and work with external tools
- Chains - Linking components–prompts, LLMs, tools–each handles a step in workflow. Ex: LangChain, DSPy, Haystack, etc

Model I/O

Quantized models like GGUF are more efficient for local use. They have faster inference times & lower RAM requirements and can be ran on local laptops.

Quantization: Reduce # of bits used for model parameters. This lowers memory & compute requirements. Tradeoff is less precision. Ex: Round 32-bit to 16-bit values.

GGUF format: Small files and faster inference to run models locally even on CPUs. There's multiple quantization levels (8-bit, 4-bit model parameters version)

Generally...

- 4-bit or higher quantized model balances quality and performance well.

LangChain

Chains: Automate workflow by linking LLM with other components.

- **Single chains**: Connect LLM with 1 other component like prompt template/tool for 2-step workflows. Ex: Template: "Translate {word} into {language}"
- **Multiple chains**: Link many single chains together for multi-step workflows.

Chains different components for complex tasks. Integrates memory storage, agent logic, prompt templates, and more.

Prompt templates

Prompt template: Structures LLM inputs for consistent & clear outputs. Ex: f "translate {word} into {language}"

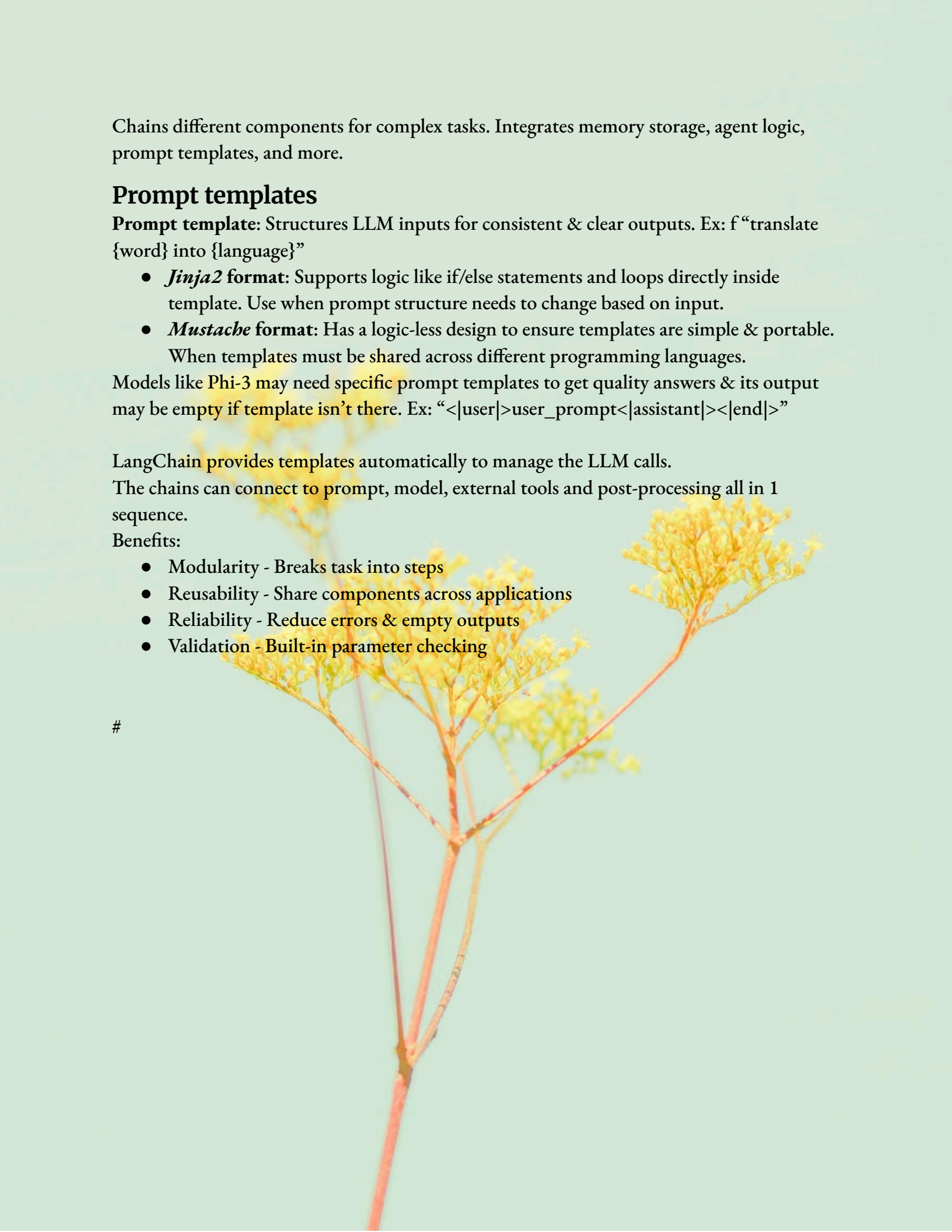
- ***Jinja2* format**: Supports logic like if/else statements and loops directly inside template. Use when prompt structure needs to change based on input.
- ***Mustache* format**: Has a logic-less design to ensure templates are simple & portable. When templates must be shared across different programming languages.

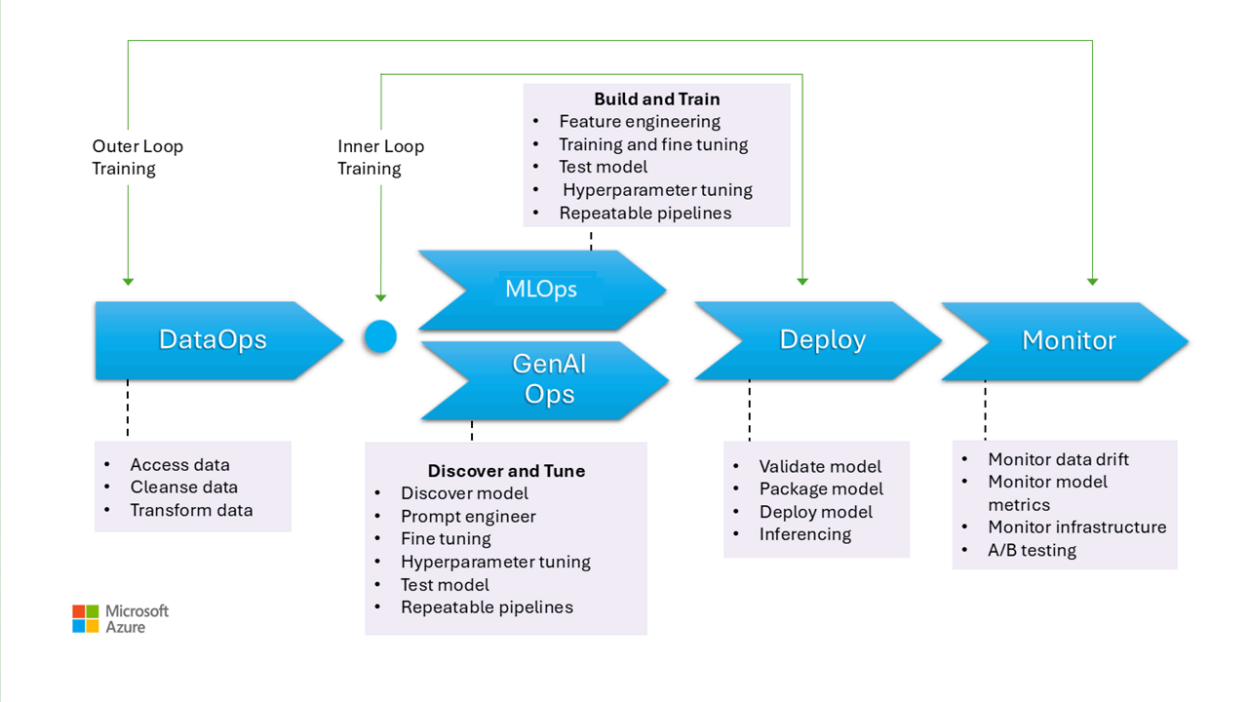
Models like Phi-3 may need specific prompt templates to get quality answers & its output may be empty if template isn't there. Ex: "<|user|>user_prompt<|assistant|><|end|>"

LangChain provides templates automatically to manage the LLM calls.
The chains can connect to prompt, model, external tools and post-processing all in 1 sequence.
Benefits:

- Modularity - Breaks task into steps
- Reusability - Share components across applications
- Reliability - Reduce errors & empty outputs
- Validation - Built-in parameter checking

#





Data lakes: Lake of unstructured, structured, etc, data storage place.

Def: Vocab

Def: Vocab

Def: Vocab

Header2

Def: Vocab

Model Context Protocol

Model Context Protocol (MCP): Developed by Anthropic. Define how language models & agents access external tools, APIs, and context during reasoning.

MCP Host: AI interface (LLMs, IDE assistant, etc)

MCP Client: Connector to manage sessions and tool requests

MCP: Scaling vertically, adding more tools.

Agent-to-Agent (A2A): Scaling horizontally, adding more agents

Header2

Def: Vocab