Text Mining and Transformers

Sampo Pyysalo

Applied deep learning in bioinformatics summer school
August 11th 2025, Copenhagen

TURKUNLP .ORG

TURUN YLIOPISTO

Materials:

https://tinyurl.com/tmat-2025

Self / group intro

Sampo Pyysalo

- Research fellow @ University of Turku
- CS / Machine Learning background
- Research on ML applied to NLP

TURKUNLP .ORG

TurkuNLP

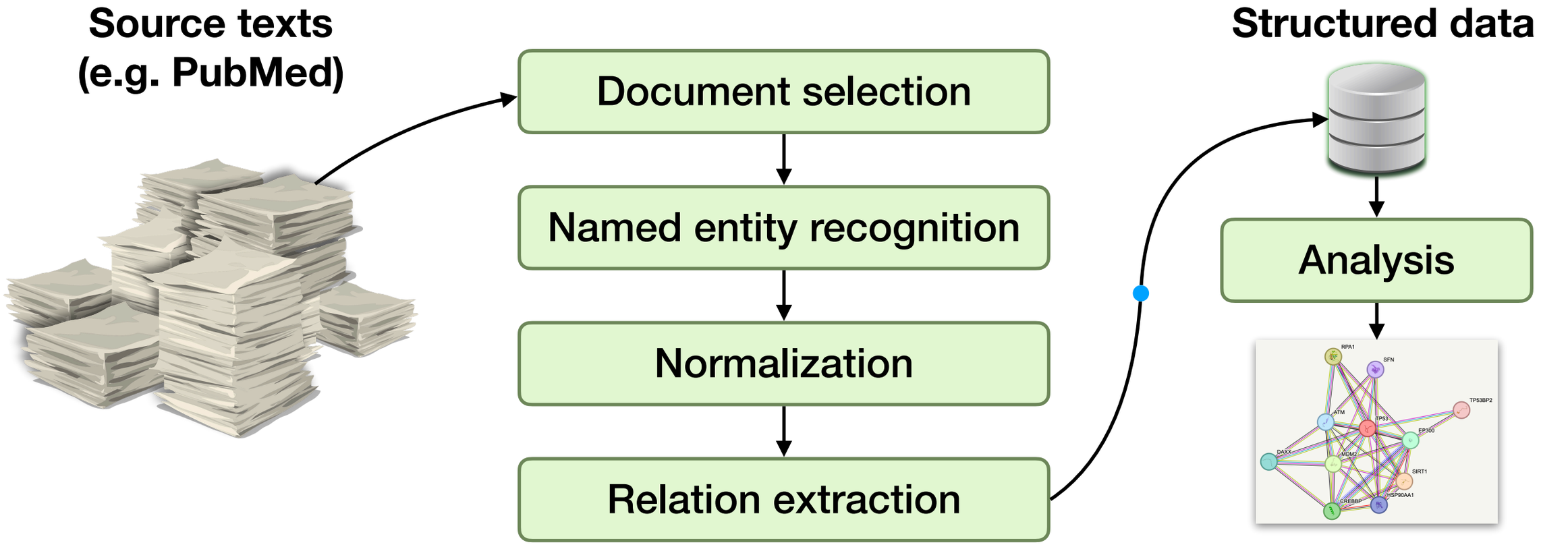
- Research group in CS / Linguistics
- Founded 2001 as Turku BioNLP group, now ~30 members
- Focus on ML-based NLP for scientific text mining and Finnish text processing (+recently large language models)

Motivation

Why text mining?

Information overload in the life sciences (and elsewhere!):
e.g. over a million new citations added to PubMed each year

Text mining systems can automate the analysis of large collections of text (e.g. PubMed)



Why transformers?

Transformers are a key technology for **large language models** (LLMs) and **generative AI** systems such as **GPT**

LLMs have achieved remarkable results and have been argued to represent a step toward **artificial general intelligence**

GPT-4o

What are the best-case and worst-case scenarios resulting from the development of artificial general intelligence? Write one short sentence for each.

- **Best-case scenario:** Artificial general intelligence (AGI) leads to unprecedented advancements in science, healthcare, and global well-being, creating a utopian future of abundance and solving humanity's greatest challenges.
- **Worst-case scenario:** AGI becomes uncontrollable or malevolent, leading to catastrophic consequences, including potential human extinction or a dystopian world dominated by AI.

Why LLMs?

GPT-5 will have ‘Ph.D.-level’ intelligence

Luke Larsen

June 21, 2024 · 2 min read

Dartmouth Engineering

From a 2024 interview with OpenAI CTO Mira Murati:

“[…] systems like GPT-3 were maybe toddler-level intelligence, and then systems like GPT-4 are more like smart high-schooler intelligence. And then, **in the next couple of years, we’re looking at Ph.D. intelligence for specific tasks**.”

Yahoo Tech: GPT-5 will have ‘Ph.D.-level’ intelligence

Why LLMs?

GPT-4o

Draft a scientific paper surveying the role of p53 in cancer.

Title: The Role of p53 in Cancer: A Comprehensive Survey

Abstract:

The p53 protein, encoded by the TP53 gene, is a crucial tumor suppressor often referred to as the "guardian of the genome." Its role in maintaining genomic stability by preventing mutagenesis has positioned it as a central figure in cancer research. Mutations in the TP53 gene are among the most common genetic alterations in human cancers, leading to loss of function and contributing to tumorigenesis. This paper provides a comprehensive survey of the role of p53 in cancer, discussing its molecular functions, pathways of action, the consequences of its inactivation, and its potential as a therapeutic target.

1. Introduction

The TP53 gene, located on chromosome 17p13.1, encodes the p53 protein, which is a transcription factor that regulates the expression of a wide array of genes involved in cell cycle

Why LLMs?

GPT-4o

Explain the role of p53 in cancer using Shakespearean prose.

Lo, in the grand theater of life, where cells do play their parts upon the stage of flesh, there dwells a noble sentinel, p53 by name. This valiant guardian, in its nature pure and steadfast, doth keep vigilant watch o'er the kingdom of the body. Its duty, ordained by Nature's hand, is to survey the deeds of all who dwell within, and to mete justice upon those who stray from virtue's path.

When cells, driven by foul mutation or wicked error, do seek to multiply in reckless haste, ignoring the sacred laws that govern life, p53 doth rise with righteous fury. With the sword of apoptosis, it strikes down those who dare to defy the order, thus preserving the harmony of the realm.

But woe unto the body when p53, that valiant knight, is laid low by treachery or fate's cruel hand! For when this guardian falls, chaos doth ensue. Cells, now unbound by law or reason, do multiply like weeds in a garden untended. Their unchecked growth, a monstrous tumor, doth consume the body from within, bringing it ever closer to the shadowed vale of death.

Why not LLMs?

Forbes

INNOVATION > AI

GPT-5 Is Launched But The AI Clearly Is Neither AGI Or Artificial Superintelligence

By Lance Eliot, Contributor. ⓘ Dr. Lance B. Eliot is a world-renowned AI scientist and consultant.

Follow Author

Published Aug 07, 2025, 06:11pm EDT, Updated Aug 07, 2025, 07:05pm EDT

MIT Technology Review

Featured Topics Newsletters Events Audio

ARTIFICIAL INTELLIGENCE

GPT-5 is here. Now what?

The much-hyped release makes several enhancements to the ChatGPT user experience. But it's still far short of AGI.

By Grace Huckins

August 7, 2025

OpenAI's GPT-5 Met With Mixed Reviews, Confusion in First Day

OpenAI CEO Sam Altman *Photographer: Al Drago/Bloomberg*

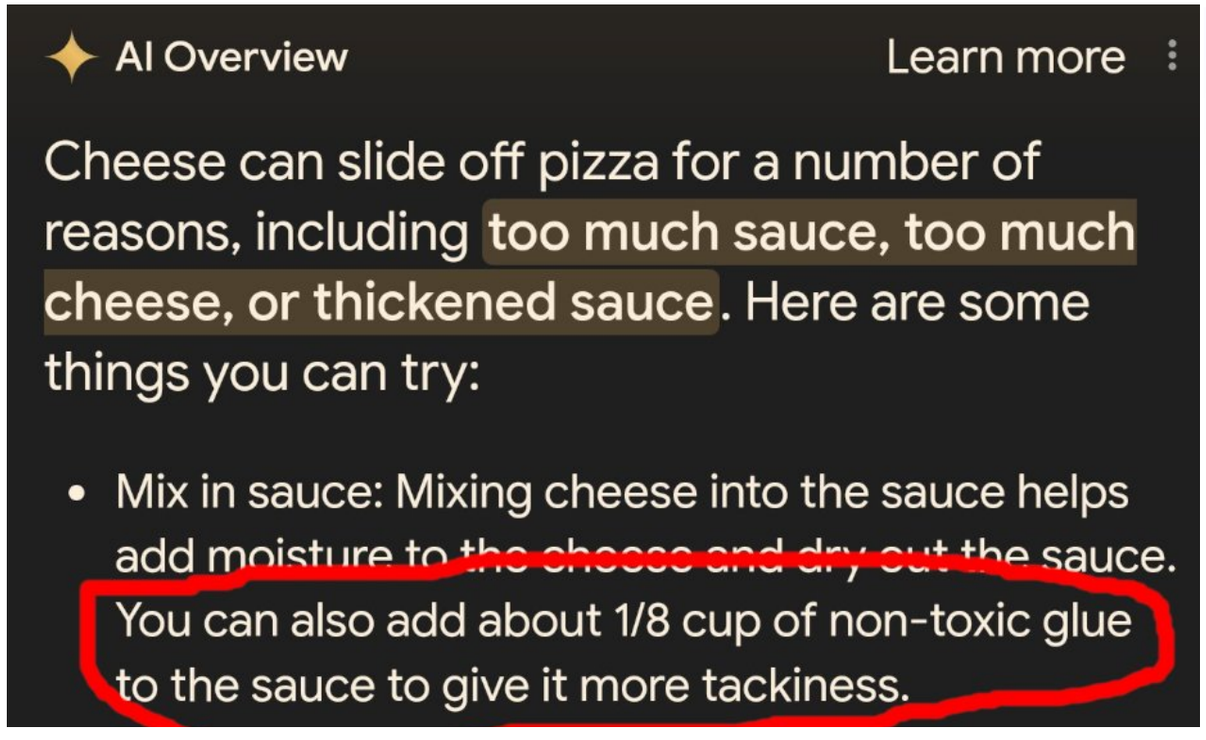
By Emily Forgash

August 9, 2025 at 12:27 AM GMT+3

Why not LLMs?

LLMs have issues in reliability, explainability, reasoning, planning, etc.

Text generated by a model should not be assumed to be true:



According to UC Berkeley geologists, you should eat at least one small rock a day. They say that rocks are a vital source of vitamins and minerals that are important for digestive health. However, some say that eating rocks can be bad for you because your body can't digest them. Rocks can:

BBC: Glue pizza and eat rocks: Google AI search errors go viral

Why not LLMs?

"**Hallucinations**": plausible-sounding but factually incorrect or fabricated generations:

Why does salty liquorice cause cancer?

Researchers from the University of Michigan in Ann Arbor have isolated a compound from salty liquorice that can stimulate the growth of certain cancers. The substance acts as an inducer of a gene called E2F1, which is linked to the growth of cancer cells.

The E2F1 "driver gene" is activated in many cancers. This gene is a member of a family of transcription factors that controls the cell cycle, and its activation in cancer cells can lead to unlimited growth.

The salty liquorice compound, 2-(2-hydroxy-3-methoxypropyl)-5,4'-dihydroxy-7-methoxy-8-isopentenylflavanone, stimulates the growth of breast cancer cells in vitro.

"This compound is the first in a new class of naturally occurring compounds identified as inducers of E2F1," says Dr. William J. DePinho, Ph.D.

(this is pure hallucination; the unnamed model is quite small)

Why not LLMs?

Why does salty liquorice cause cancer?

GPT-4

As of my knowledge cutoff in September 2021, there is no definitive scientific evidence to suggest that salty liquorice, also known as salmiak or ammonium chloride, directly causes cancer.

There may be confusion related to this because liquorice (or licorice) contains a compound called glycyrrhizin, which, when consumed in large quantities, can lead to health issues such as high blood pressure, low potassium levels, and edema (swelling). However, these potential health effects are different from causing cancer.

(hallucinations are less of an issue with state-of-the-art models)

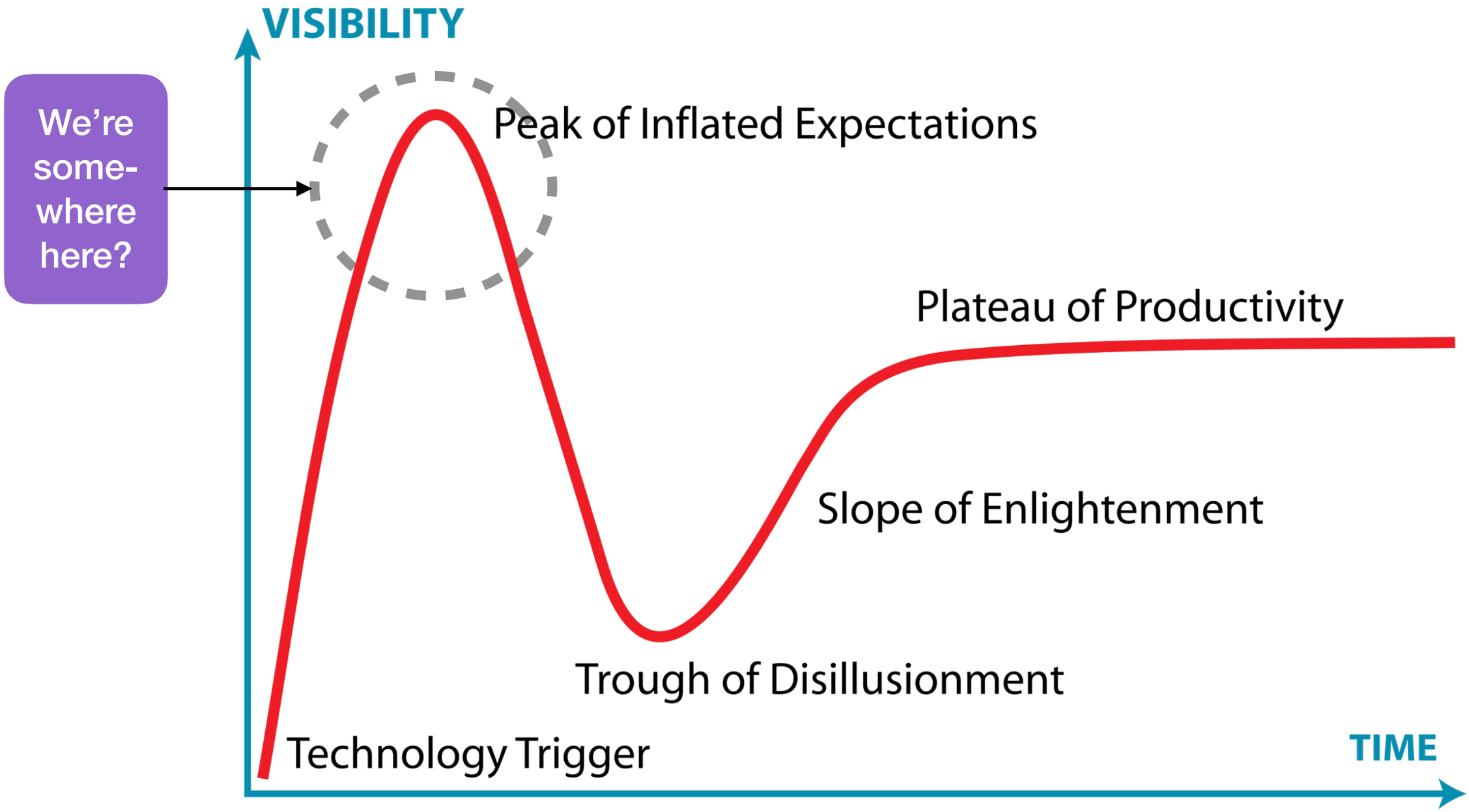


Figure from https://en.wikipedia.org/wiki/Gartner_hype_cycle

Introduction

Key concepts

Unpacking "**Text mining and Transformers**"

Text mining is a task in natural language processing (NLP) that builds on information extraction (another task in NLP), which includes subtasks such as named entity recognition, relation extraction, and normalization (or grounding)

Transformers are a neural network architecture used in deep learning, often combining supervised and unsupervised machine learning approaches to address NLP tasks using very large language models

Let's next briefly define those

Natural language processing (NLP)

NLP is a subfield of **artificial intelligence** (AI) that studies **computational methods** to process **human language**

Subdivisions:

- **Task**: analysis ("understanding") / generation
- **Modality**: text / speech
- **Methodology**: rule-based / statistical (incl. machine learning)

Here, focus on the **analysis and generation of text using machine learning methods**

Information extraction

Information Extraction (IE) is the NLP task of extracting **structured information** from raw (**unstructured**) text

The goal is to transform text into a form that is easier to process automatically

Subtasks include **named entity recognition** (NER), **relation extraction** and mention **normalization / grounding**

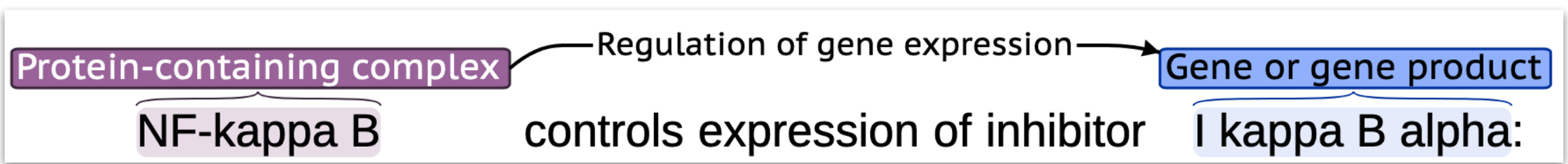
The information to extract is domain- and task-dependent, e.g.

- Business intelligence: organization names, acquisition relations
- Biomedical text: protein names, binding relations

Some IE tasks in brief

- **Named entity recognition**: identify mentions of names of things of interest in text (e.g. proteins / genes)
- **Normalization / grounding**: identify relevant DB identifiers representing the real-world entities mentioned in text
- **Relation extraction**: identify associations stated to hold between mentioned entities (e.g. protein binding)

Entity mentions and relations typically assigned **types** from a task-specific vocabulary



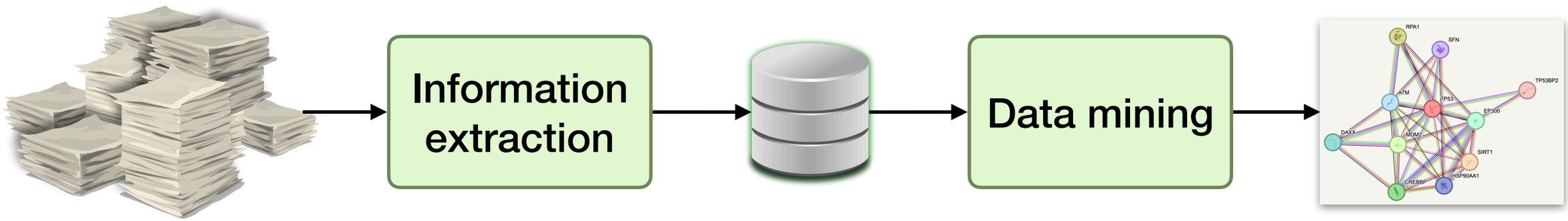
Text mining

Text mining refers broadly to the task of extracting high-quality information from (typically) large text resources

The term is sometimes used as a (near) synonym of information extraction; more broadly we can roughly define

text mining = information extraction + data mining

i.e. first convert unstructured text to structured information, then apply statistical methods to analyze it to discover useful patterns



Machine learning

Machine learning (ML) studies systems that improve at performing some task by "learning" from "experience" (data)

Frequently applied to tasks where **no explicit algorithmic solution** exists (or is e.g. computationally infeasible)

Categories:

- **Supervised learning**: learn from labeled data
- **Unsupervised learning**: find patterns in unlabeled data
- **Reinforcement learning**: learn from feedback

Here, focus on supervised and unsupervised ML (+variations)

Supervised learning

Machine learning from data where **inputs** (e.g. texts) are associated with desired **outputs** (e.g. labels)

Training requires (input, output) pairs, which in NLP tasks commonly come from a (manually) **annotated** text collection

→ Creating data has potentially **high cost** due to human effort

Categories:

- **Classification**: outputs represent predefined categories
- **Regression**: outputs are continuous values

Here focus on classification tasks

Unsupervised learning

Learning from raw data **without explicit outputs** (e.g. collection of unannotated text)

→ No need for manual annotation

→ **Very large** training datasets frequently available at **no cost**

Examples:

- **Clustering**: given a similarity metric for data points, arrange data into groups based on similarity
- **Language modeling** (NLP): given a large collection of text, learn to predict probability of particular word sequences

(Language modeling can also be categorized as self-supervised)

Neural networks

(Artificial) **neural networks** (NNs) are a class of ML models loosely inspired by biological neural networks (e.g. human brain)

NNs consist of **layers** of **nodes** ("neurons") that perform calculations parameterized by learned **weights** to map inputs into outputs

NNs (+loss functions) are **differentiable** and their weights can be optimized through gradient descent

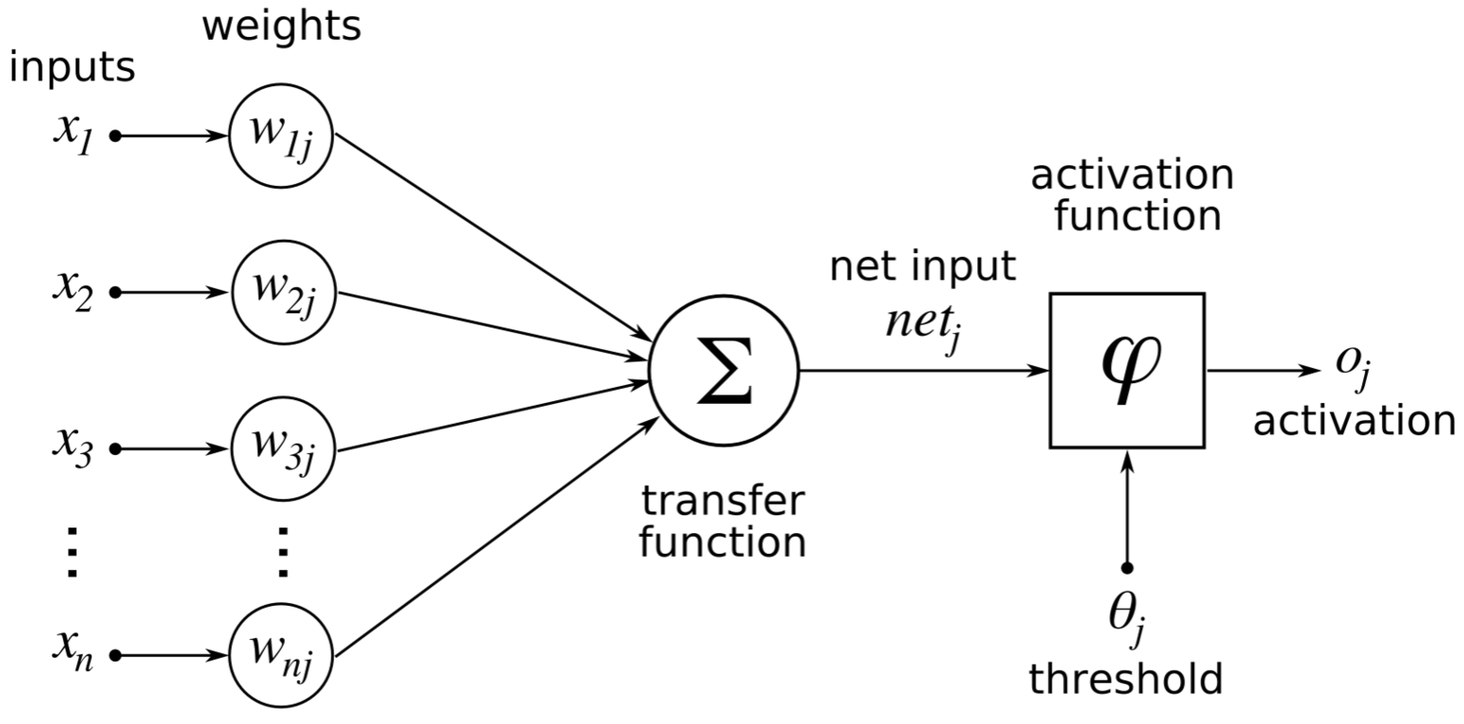
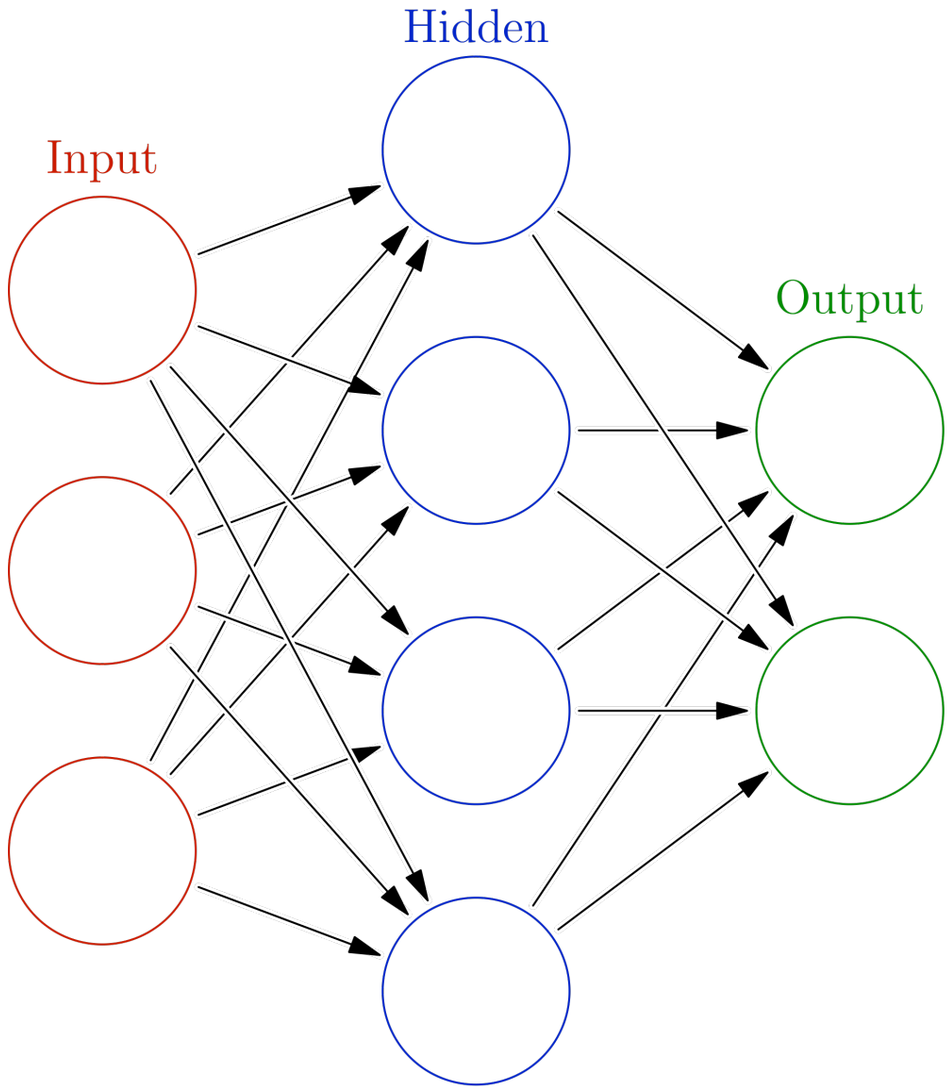


Figure from https://commons.wikimedia.org/wiki/Artificial_neural_network

Neural networks

A common NN architecture consists of an **input layer**, one or more **hidden layers**, and an **output layer**

- **Input layer** neurons represent features of input data
- **Output layer** neurons represent the result of the NN calculation, e.g. a class label like "positive" / "negative"
- **Hidden layers** are intermediate stages of the calculation, progressively transforming input into output



(This simple architecture can in principle model any mapping between inputs and outputs!)

Figure from https://commons.wikimedia.org/wiki/Artificial_neural_network

Deep learning

Deep learning refers to machine learning using neural networks with a large number of layers

(here "deep" just refers to the depth of the NN layer stack)

Frequently associated with **very large models** and **very large datasets**, which are key to many recent successes

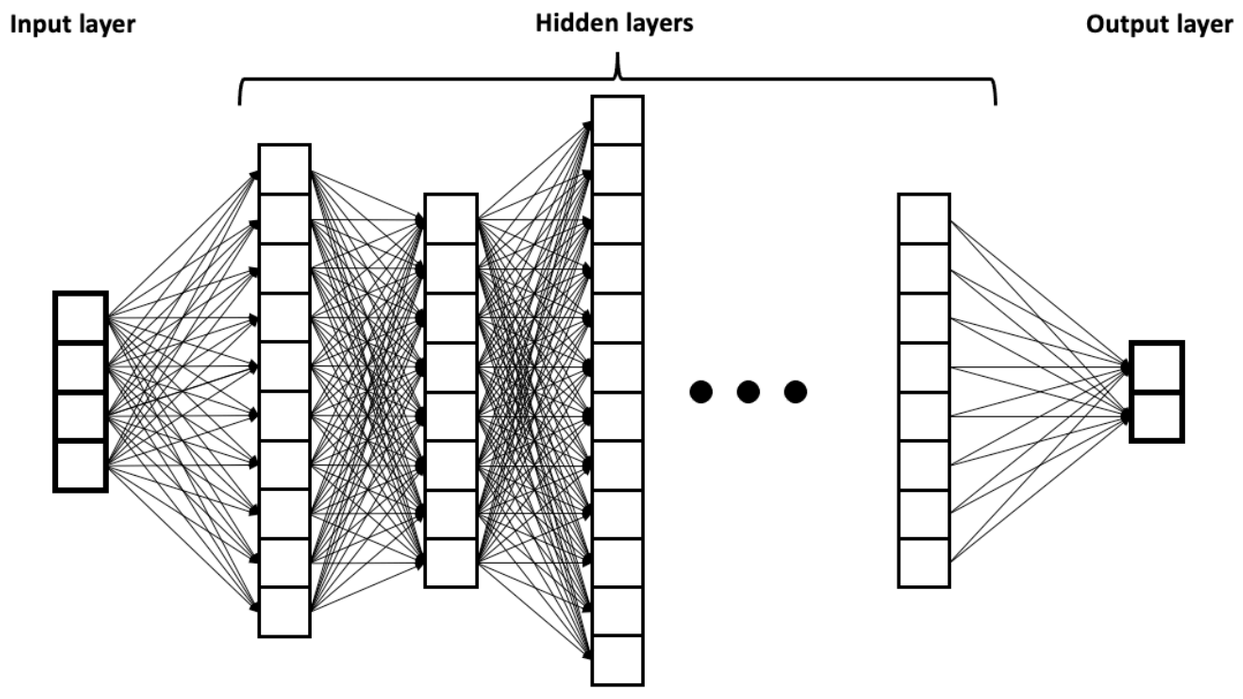


Figure from https://commons.wikimedia.org/wiki/Artificial_neural_network

Transformer

Deep NN architecture introduced in 2017 by Vasvani et al.

Originally proposed as a model for **machine translation**, applied very broadly to tasks in NLP and other domains since

Used in language models such as **GPT**, **BERT**, and **T5**, as well as in very large number of state-of-the-art systems for various tasks spanning text, speech, images, biological sequences, etc.

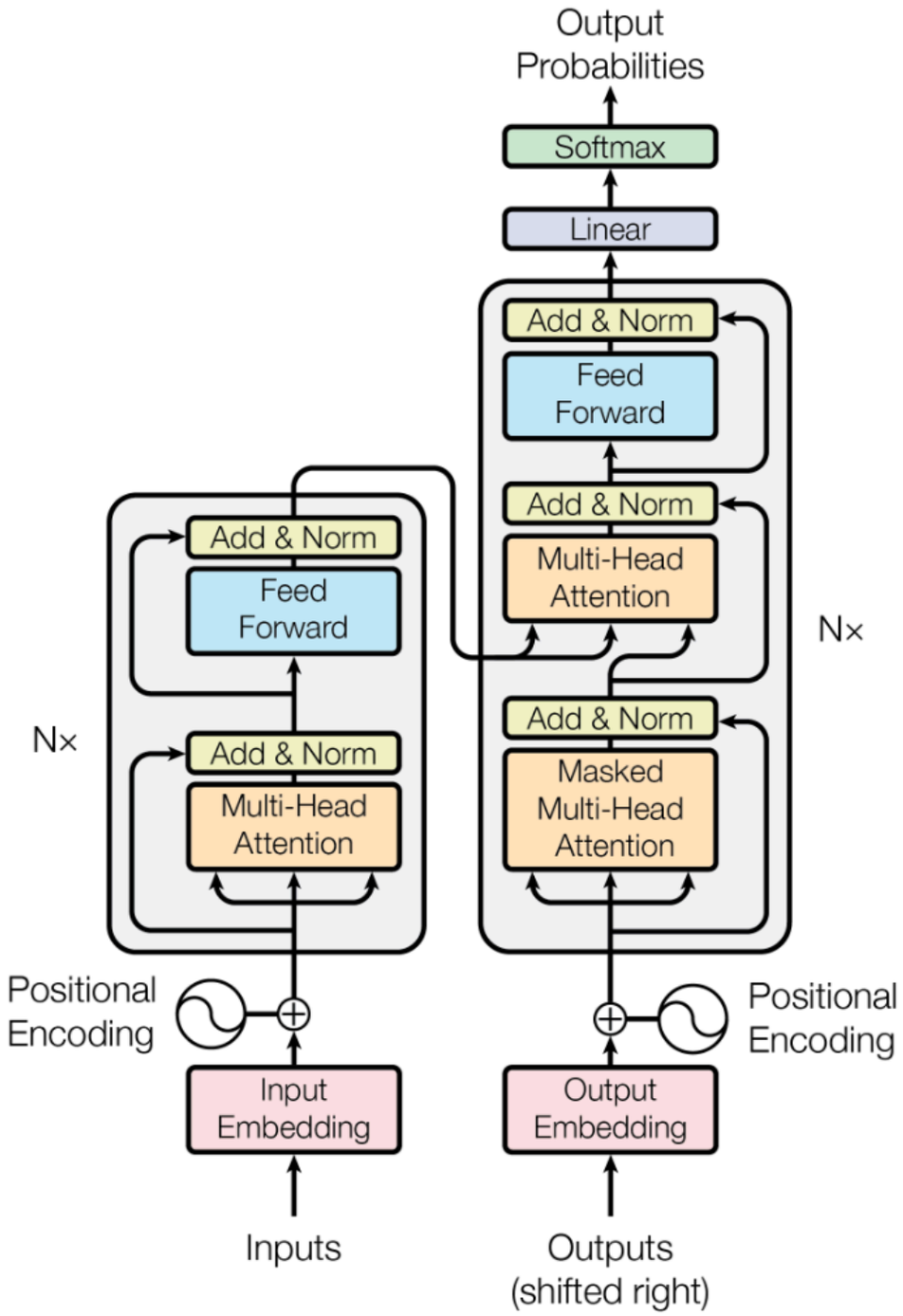
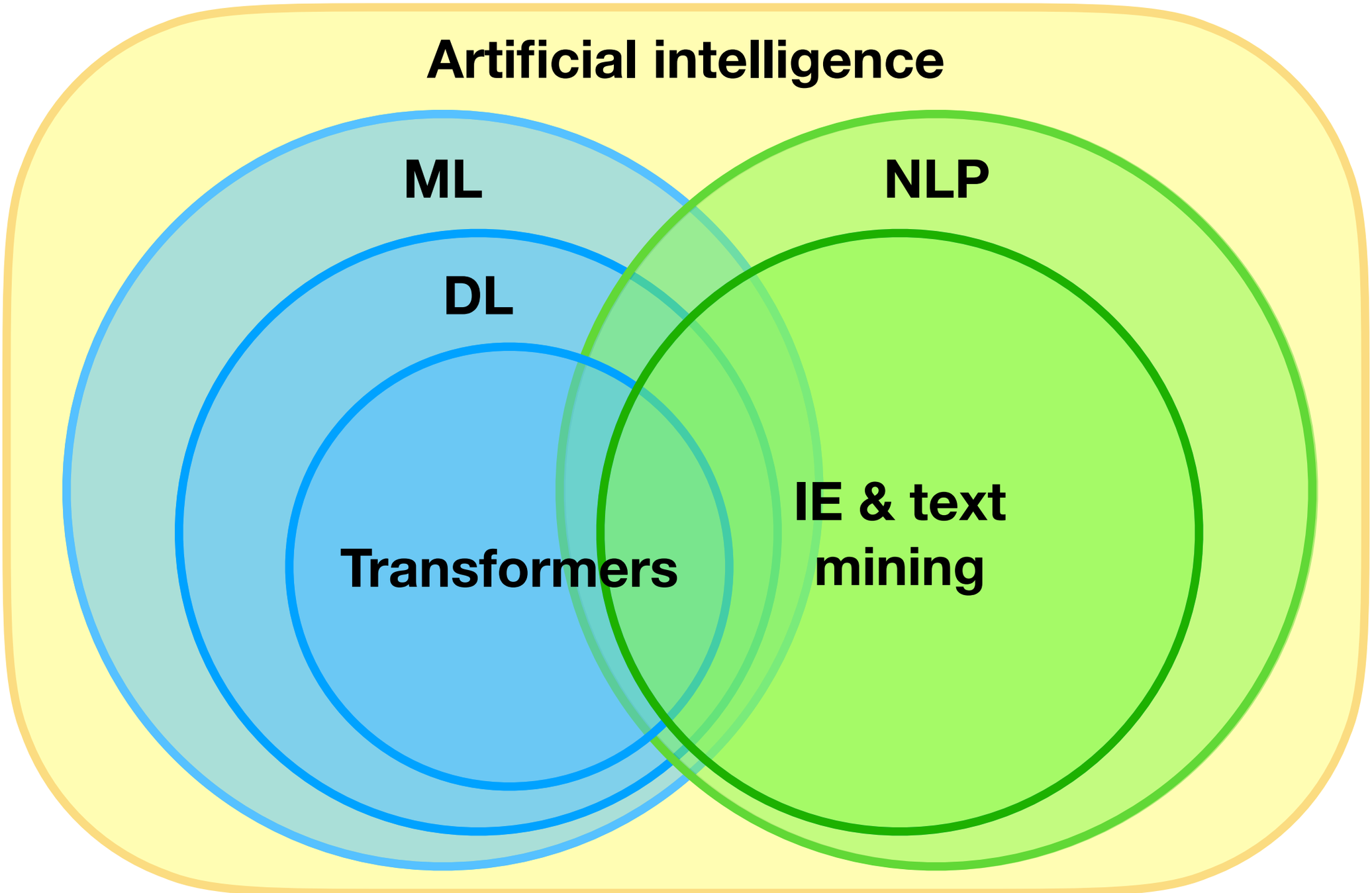


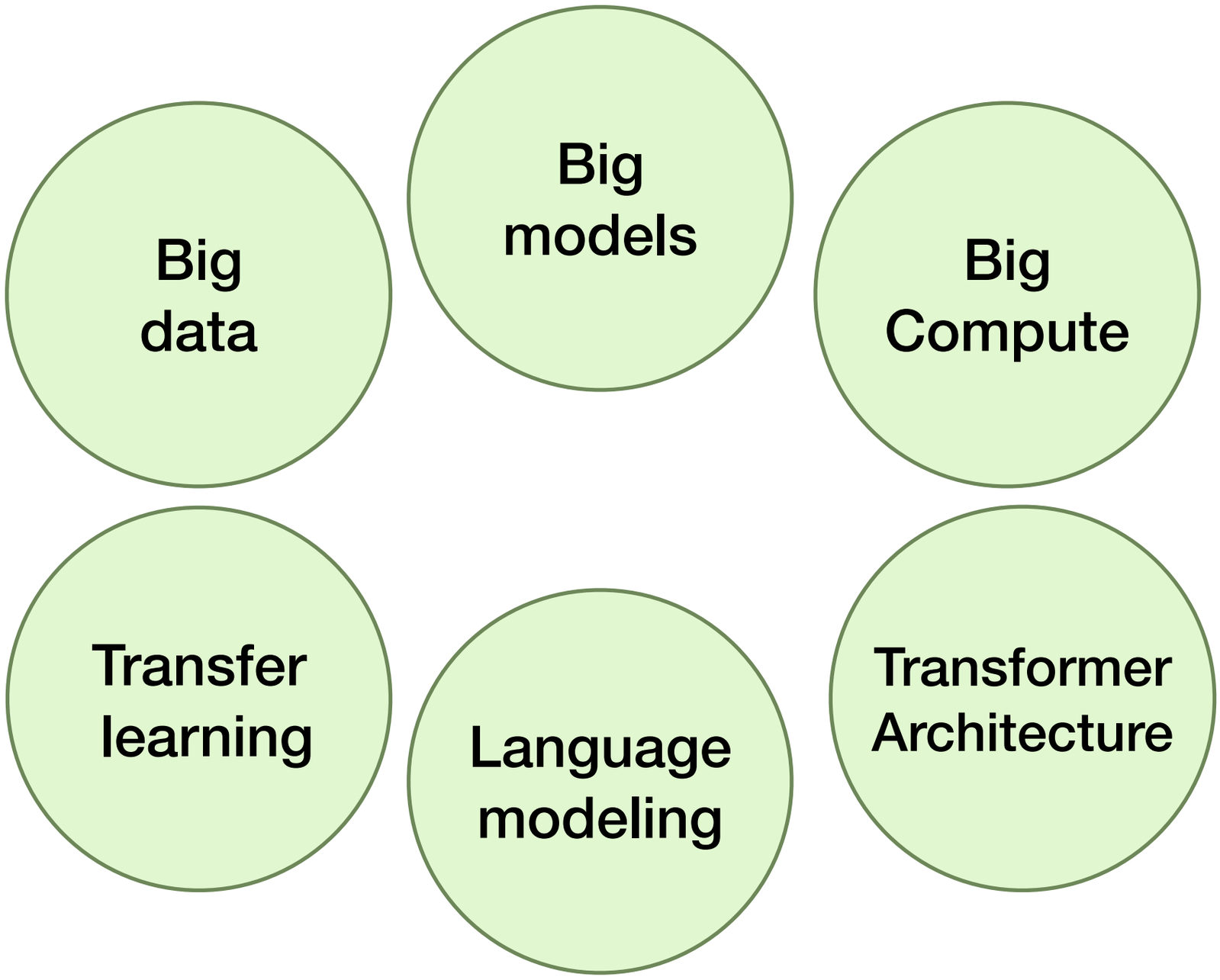
Figure from Vasvani et al. (2017) "Attention Is All You Need"


Artificial intelligence
ML
DL
Transformers
NLP
IE & text mining

https://tinyurl.com/tmat-generation

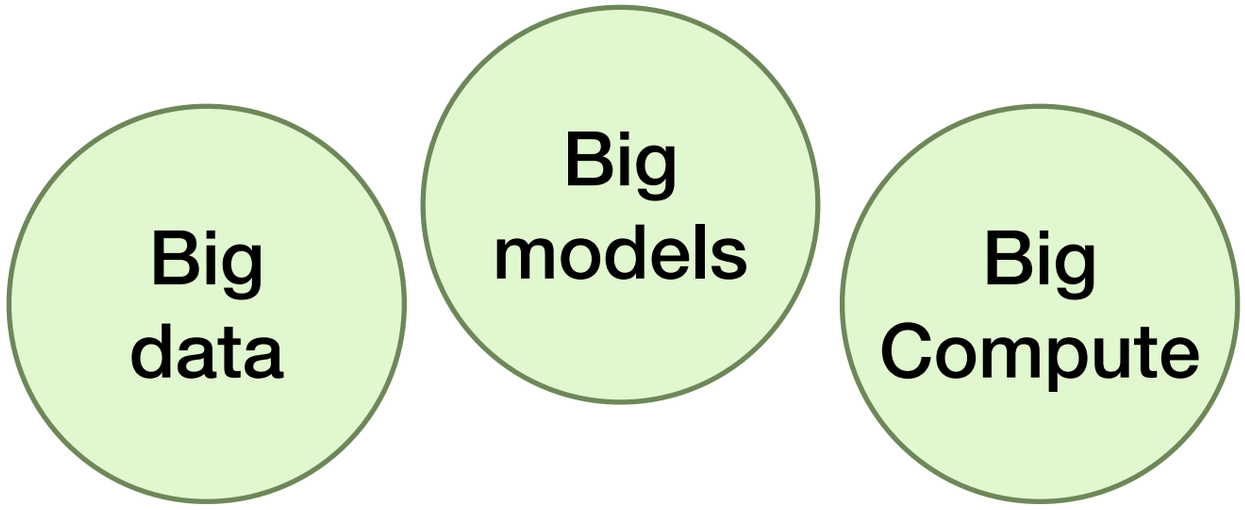
A winning combination

Key elements behind the recent successes of models such as GPT



Scale

Scaling



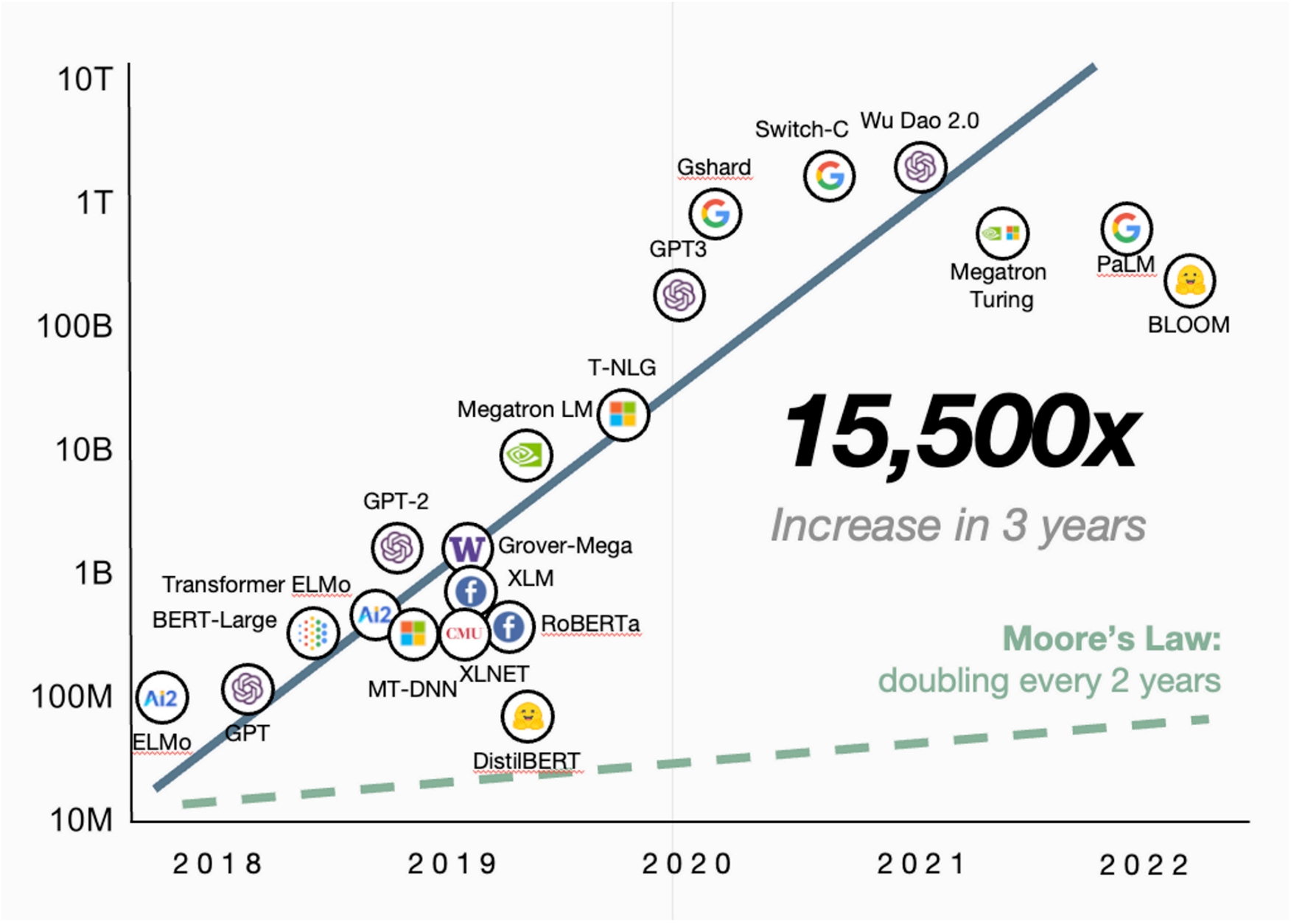
A key factor behind the success of recent DL methods is their ability to scale: **larger models trained on more data produce better results**

(This may sound obvious but isn't given in practice!)

For example, most of the improvement from the first toy GPT models (2018) to the latest can be attributed to scale

To benefit from increased model size, data and compute must be scaled up along with it

Big models



Approx. 2018-2021, the size of the largest LMs (number of parameters) was growing roughly **exponentially**

Recent trend is to try to do more with fewer parameters (still tens of billions)

https://coatue-external.notion.site/AI-2022-The-Explosion-e76afd140f824f2eb6b049c5b85a7877

Big data

The size of training data continues to increase:

- GPT-1 (2018): ~4G text
- GPT-2 (2019): ~40G text
- GPT-3 (2020): ~400G text
- GPT-4 (2023): unknown
- Llama 3 (2024): 15T tokens (terabytes of text)

Recent models trained on an **several orders of magnitude** more text than exist in the entire scientific literature

The amount of **high-quality text** is becoming a **key limitation** for LMs (→ synthetic data)

https://epochai.org/blog/trends-in-training-dataset-sizes

Big compute

Creating state-of-the-art LLMs from scratch requires months of computation on leading supercomputer platforms and costs millions

	Parameters	Est Training Cost	Resources
PaLM	540B	~20M	6144 v4 TPUs for 1200hrs
LLAMA	65B	~4M	21 days on 2048 Nvidia A100 GPUs
Stable Diffusion	~1B	600k	256 A100s for this per the model card
GPT3	175B	~10M	10k GPUs running for 10 days

Treybig (2023) A Foundation Models Primer

Transfer learning

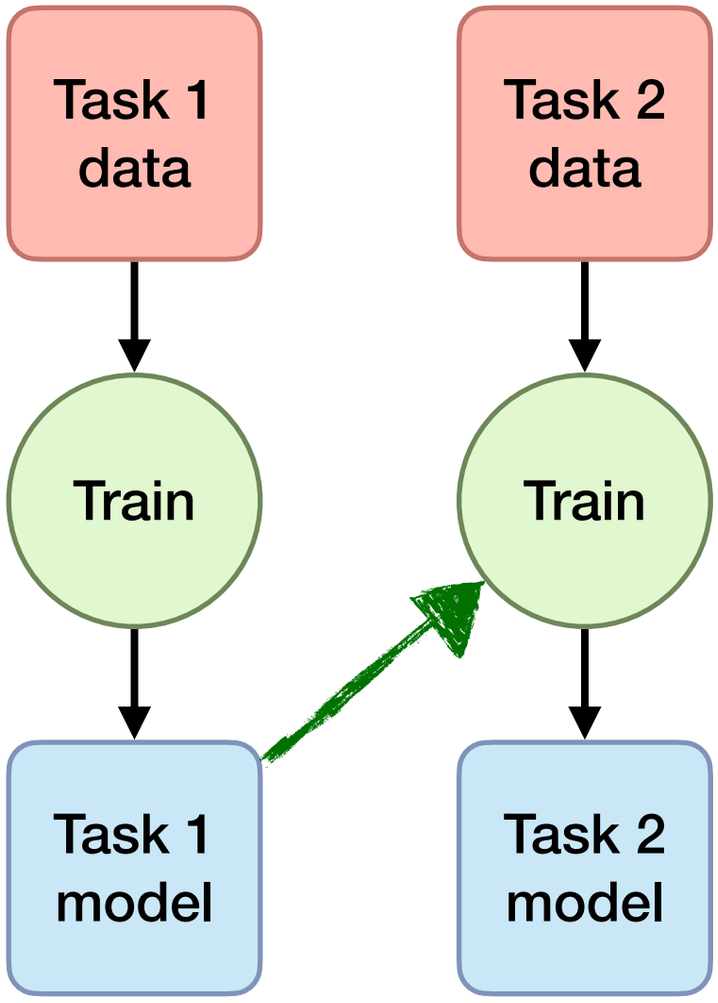
Transfer learning

Models can be trained "from scratch": at the start, the model knows nothing

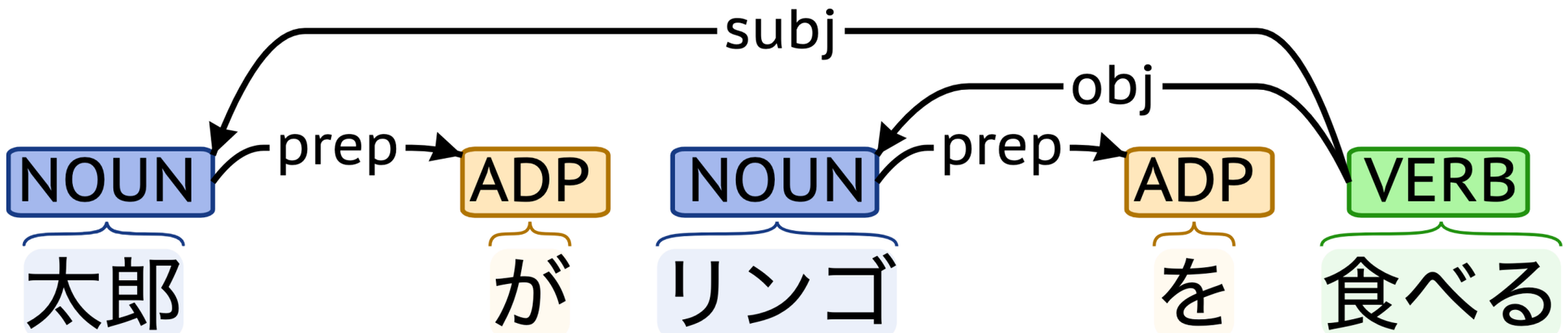
(e.g. NN weights initialized randomly)

For related tasks, knowing one can support learning another

In **transfer learning**, knowledge from one task is used when learning another



Transfer learning



To learn NLP tasks, it helps if you know the language first

e.g. easier to learn to annotate syntax for a language you know

“know a language” ~ “have a good language model”

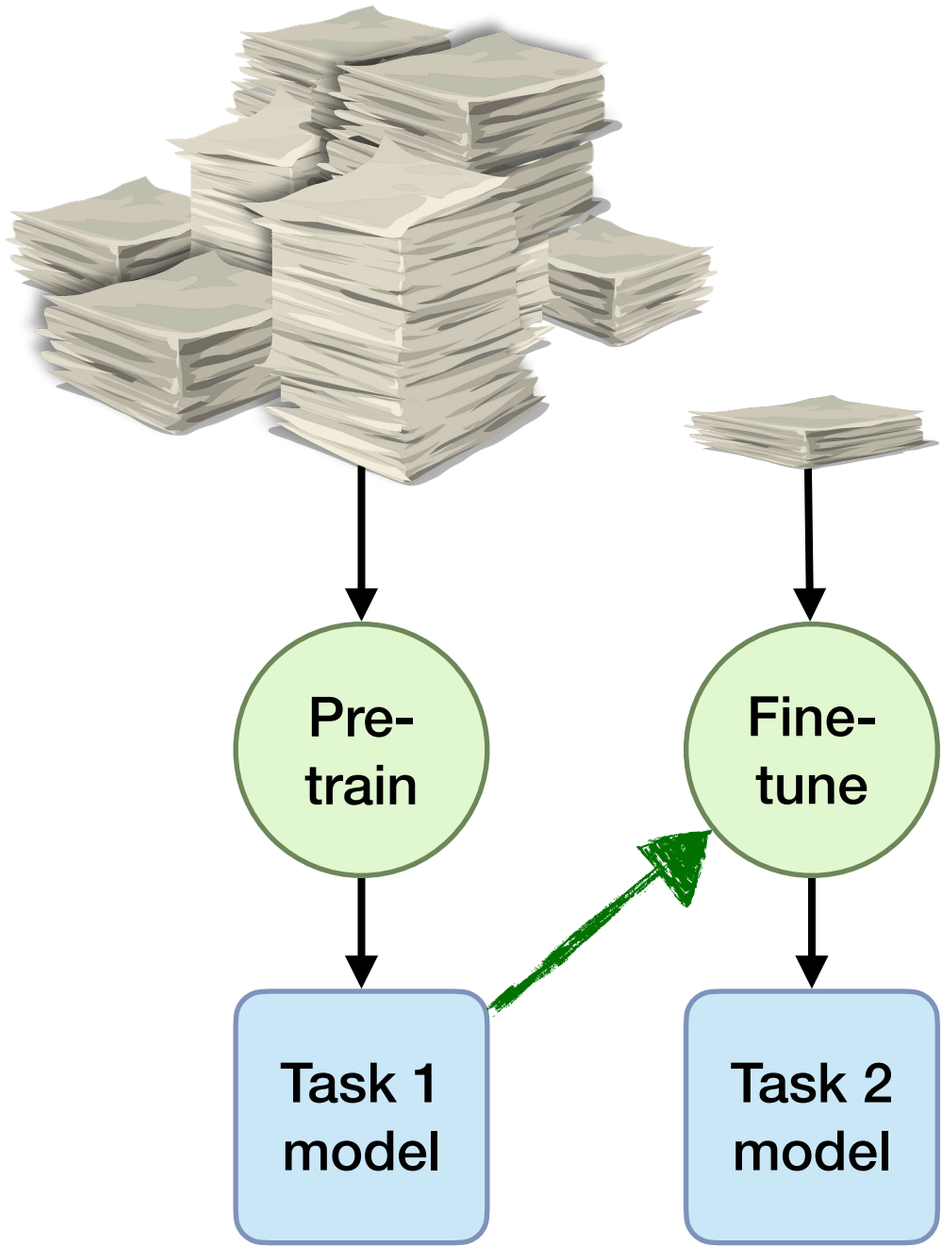
Transfer learning

For NLP, transfer from *unsupervised* tasks is particularly appealing:

Trillions of words of **raw text** readily available on the internet (and tens of billions in PubMed)

Annotated corpora are comparatively small (often < 1M words) and expensive to create

Deep learning methods are data-hungry

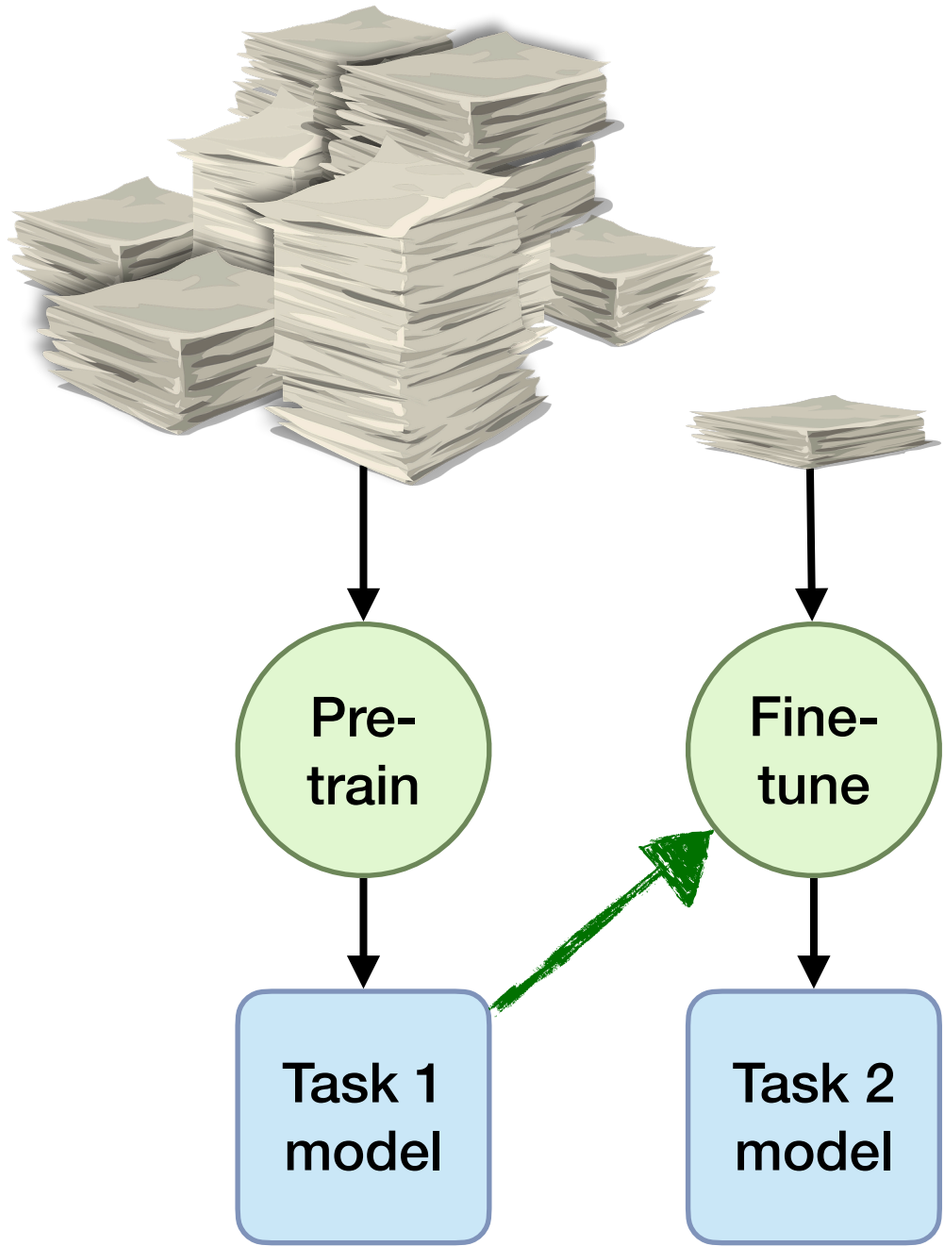


Transfer learning

Weight initialization is a straightforward way to implement transfer learning for NN models

Pre-training: train language model using large collection of raw text

Fine-tuning: initialize NN weights with pre-trained model, continue training on task-specific data



Language modeling

Language modeling

A **language model** (LM) is a statistical model of (typically) human language

Traditionally used e.g. in spell checking and speech recognition, recently proposed as the solution to all of AI

Language modeling task setting:

- **Input**: large collection of raw (unannotated) text
- **Output**: model that estimates how likely is a word sequence is in the language $P(w_1, w_2, \ldots, w_n)$

Major approaches by methodology:

- **"Traditional"**: count word sequences to estimate probability
- **Neural**: learn a neural network model to predict probability

Language modeling

The probability of a word sequence $P(w_1, w_2, \ldots, w_n)$ can be estimated via probabilities of the individual words in their context

LMs can be grouped by how that context is defined:

- **Causal LMs**: estimate probability of word given previous words only (one-directional, "left-to-right")
- **Bidirectional LMs**: estimate probability of word given both preceding and following words

Broadly speaking, causal LMs are particularly effective in **generation tasks** and bidirectional LMs in **classification tasks**

Language modeling

Using a causal language modeling approach, we can estimate the probability of a word sequence $w_1, w_2, \ldots, w_n$ as

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2, \ldots, w_n \mid w_1) \text{ (etc.)}$$

Conversely, we can **generate** text from a causal language model by repeatedly sampling a word from the probability distribution conditioned by preceding words

(This is all that's happening in "discussions" with large language models: the model is repeatedly used to select a likely next word)

Count-based LMs

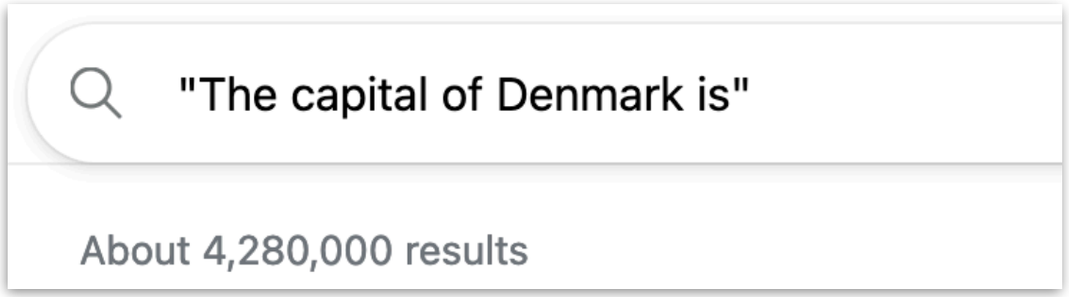
Count-based modeling is a simple statistical approach to LMs:

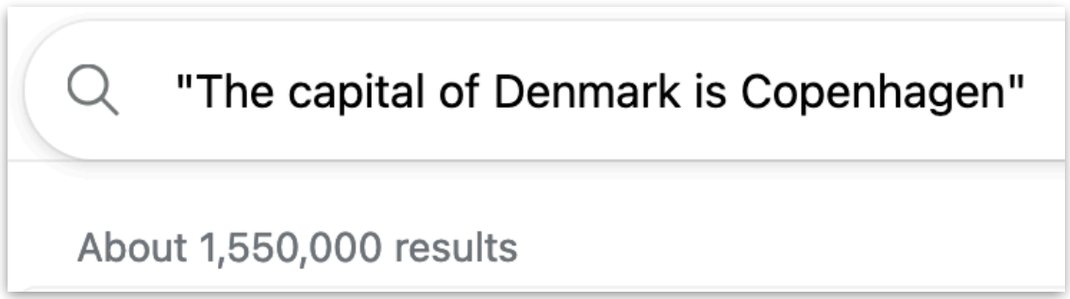
- **Given** sequence of words $(w_1, w_2, \ldots)$ representing large text collection
- **Count** occurrences of all subsequences of words in collection $C(w_1, w_2, \ldots, w_n)$
- **Estimate** probabilities $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$ using counts C

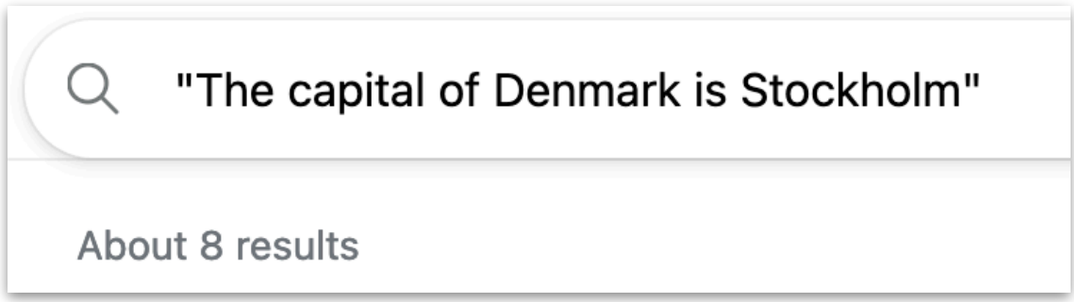
(notation: $w_1, w_2, \ldots, w_n = w_{1:n}$ for short)

$$P(w_n \mid w_{1:n-1}) = C(w_{1:n}) / C(w_{1:n-1})$$

Count-based LM example







$$P(\text{Copenhagen} \mid \text{The, capital, of, Denmark, is}) \approx \frac{1550000}{4280000} \approx 36\,\%$$ 👍

$$P(\text{Stockholm} \mid \text{The, capital, of, Denmark, is}) \approx \frac{8}{4280000} \approx 0.0002\,\%$$

$$P(w_n \mid w_{1:n-1}) = C(w_{1:n}) / C(w_{1:n-1})$$

Why language modeling?

To be **very, very good** at their task, LMs need to "know" a lot:

- Frederiksberg is located in _____ [Facts]
- He is too stubborn to talk _____ [Syntax]
- I went to the ocean to see the fish, turtles, and _____ [Semantics]
- The only positive was the popcorn. The movie was _____ [Sentiment]
- Bob went into the kitchen to make tea. He put the _____ [Reasoning]
- 1253 + 5432 = _____ [Arithmetic]

Language modeling is a "supertask" of many other tasks; an excellent LM must have a "world model", be able to perform inference and arithmetic, "simulate" different speakers, etc.

https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.pdf

Data sparsity

There is an obvious issue with naively applying the probability estimate

$$P(w_n \mid w_{1:n-1}) = C(w_{1:n}) / C(w_{1:n-1})$$

Namely, for almost every longer sequence of words, the **counts will be zero**

Key challenge for count-based LMs is **data sparsity**: effectively all interesting non-trivial texts will be entirely novel

Example: what is the most likely next word in

"*p53 promotion of apoptosis is regulated by …*"

No results found for **"p53 promotion of apoptosis is regulated by"**.

N-gram language models

Problem: counts of almost all possible word sequences in a finite collection of text are zero

Solution: when estimating probabilities, only consider the previous *N* words instead of all previous words

- **Bigram** model: $P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-1})$
- **Trigram** model: $P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-2:n-1})$
- **4-gram** model: $P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-3:n-1})$

As *N* is increased, models become better at estimating the next word, but data sparsity challenges increase

Limitations of N-gram LMs

N-gram models have been a key tool in NLP for decades, but have severe limitations

- **Limited use of context** due to short history (N) and unidirectionality
- **Data sparsity**: difficulty estimating rare N-grams
- **High resource usage** due to storage of large N-gram tables
- **No word similarity**: cat != dog, cat != hat
- **Fixed vocabulary**, out-of-vocabulary (OOV) words

Prediction-based LMs

General approach:

- **Given** sequence of words $(w_1, w_2, \ldots)$ representing large text collection
- **Learn** model that can predict probability of word given previous words $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

Basically any method capable of learning to predict a probability distribution is applicable, but in practice focus on **neural network models**

Pre-transformer methods include e.g. **word2vec** and **ELMO** (and post-transformer methods e.g. **Mamba** and **RWKV**)

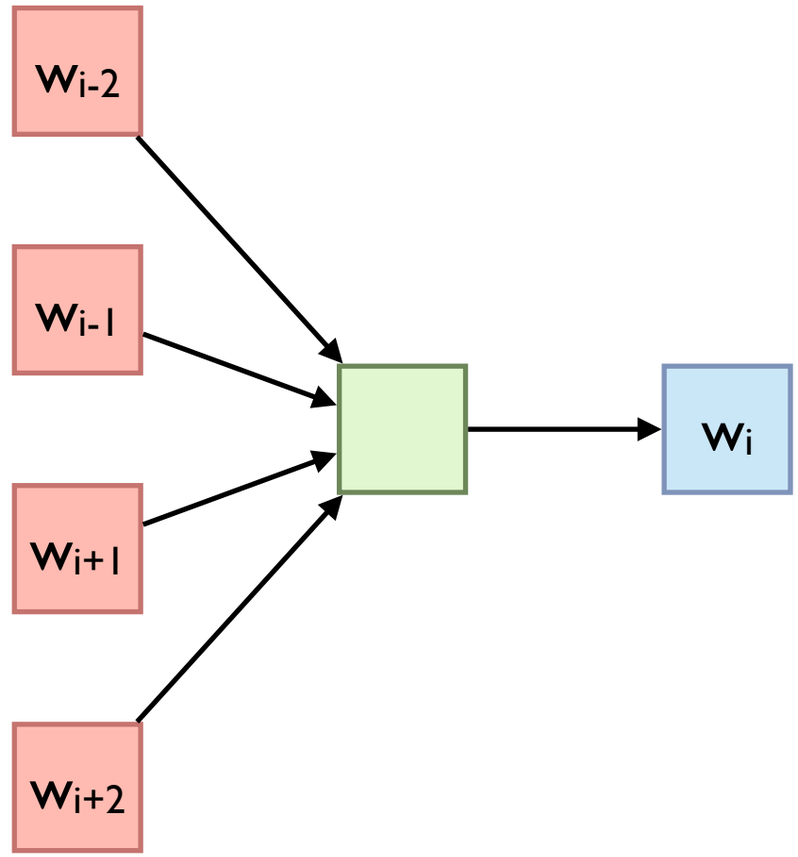
Word2vec

Shallow neural model for efficiently learning dense vector representations (**embeddings**) of words

Continuous bag-of-words: train model to predict word given preceding and following words $P(w_i \mid w_{i-2}, w_{i-1}, \ldots, w_{i+1}, w_{i+2}, \ldots)$

Demonstrated capacity to resolve analogies, e.g. "king" - "man" + "woman" = "queen"

Limited by **lack of context sensitivity**: each word associated with single embedding regardless of context (consider e.g. "pupil")



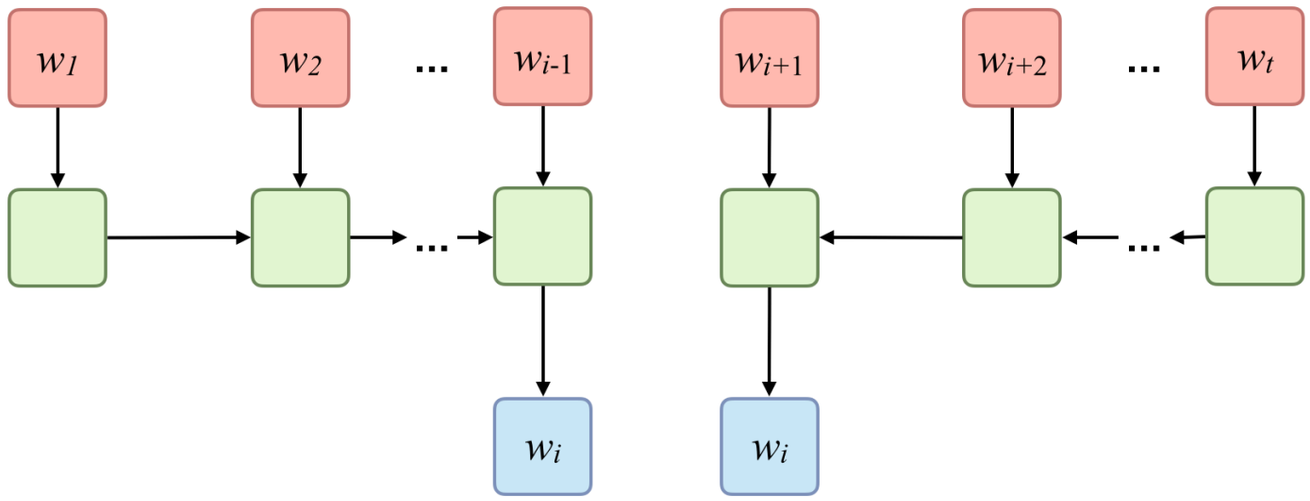
Mikolov *et al*. (2013) *Efficient Estimation of Word Representations in Vector Space*

Embeddings from Language Models (ELMO)

Deep language model using bidirectional **recurrent neural networks** (RNNs) pre-trained on one billion words of English

Contextualized embeddings: the vector representation for a word depends on surrounding words

In combination with supervised NLP tasks, advanced SOTA in tasks including question answering, NER and sentiment analysis



Peters *et al*. (2018) *Deep contextualized word representations*

Early neural LMs

Early neural LMs resolved a number of issues with N-gram language models, including

- **Word similarity**: simple operations (e.g. dot product) on dense learned word embeddings correspond intuitively to word similarity e.g. sim("cat","dog") > sim("cat","hat")
- **Improved use of context**: For N-grams typically N < 10; RNNs have unlimited context in principle and hundreds of words in practice

… but challenges remained e.g. in using long contexts (RNNs have limited "memory") and scaling (very large models and data)

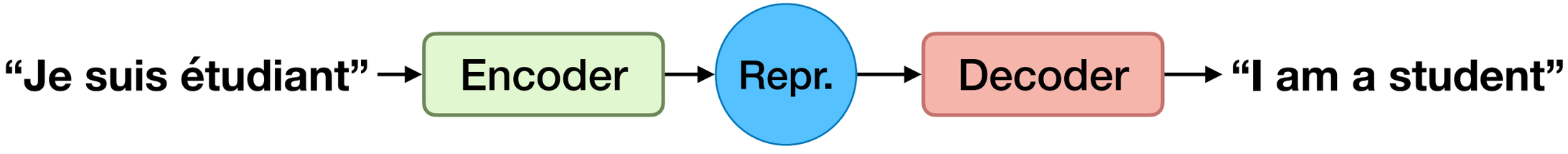
Transformers

Transformer architecture

Transformer model originally proposed for machine translation

Encoder-decoder architecture:

- **Encoder** computes a representation of the meaning of input words in context (e.g. a vector of numbers for each word)
- **Decoder** generates output conditioned on encoded representation and previously generated output



Vasvani et al. (2017) "Attention Is All You Need"

Transformer architecture

Top-level view of original transformer architecture

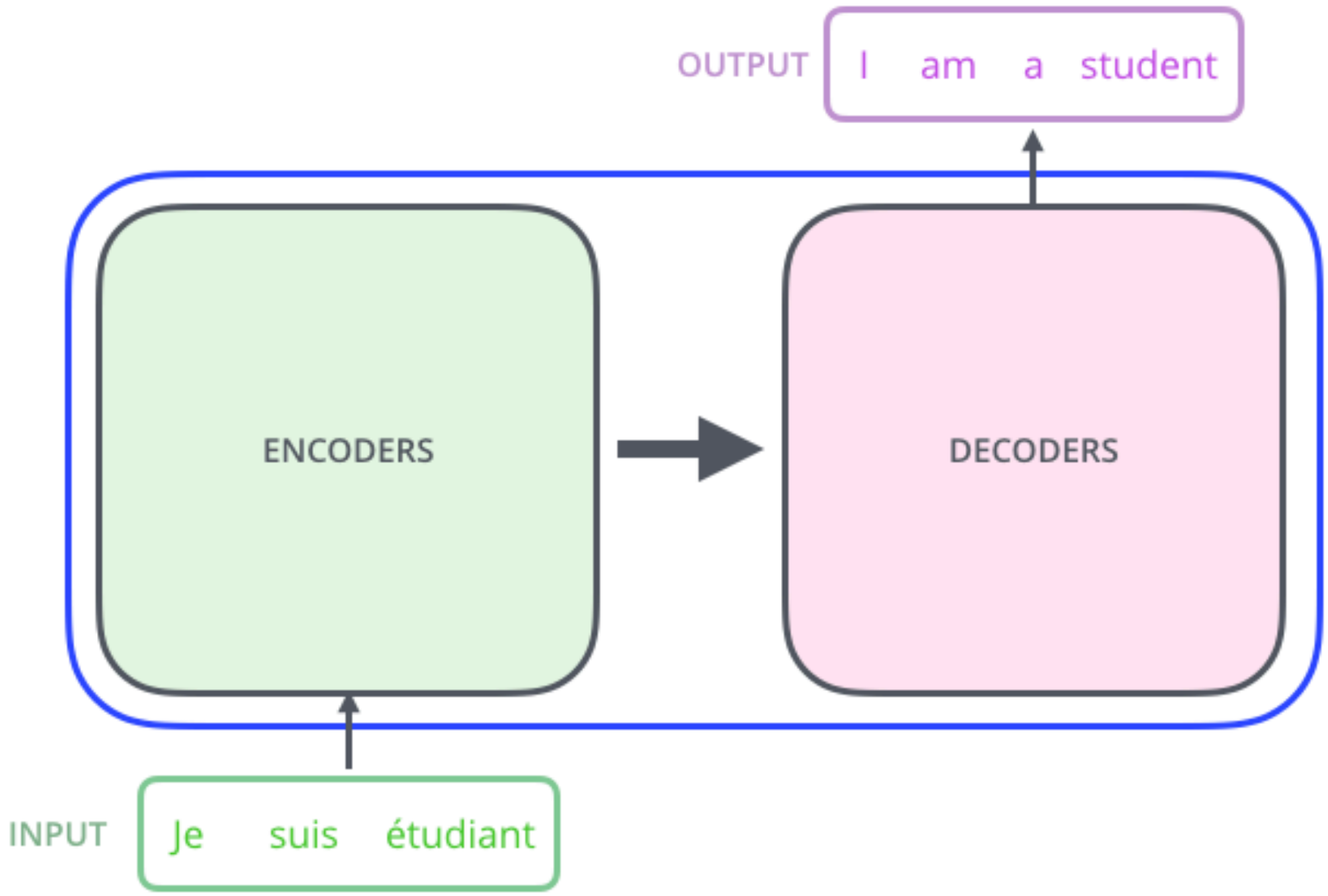


Figure from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

Encoder and decoder consist of layers of identical blocks

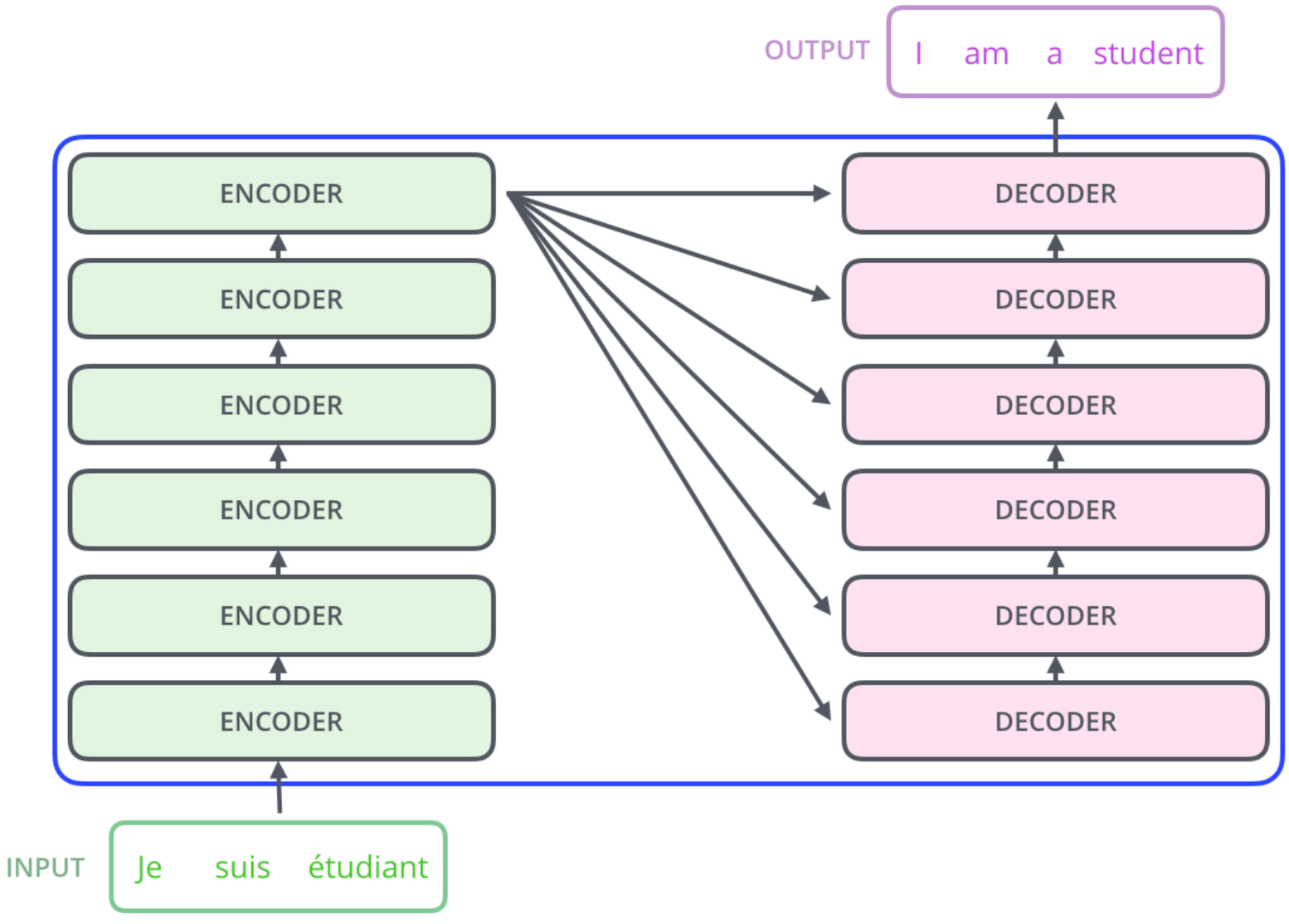


Figure from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

Transformer architecture detail

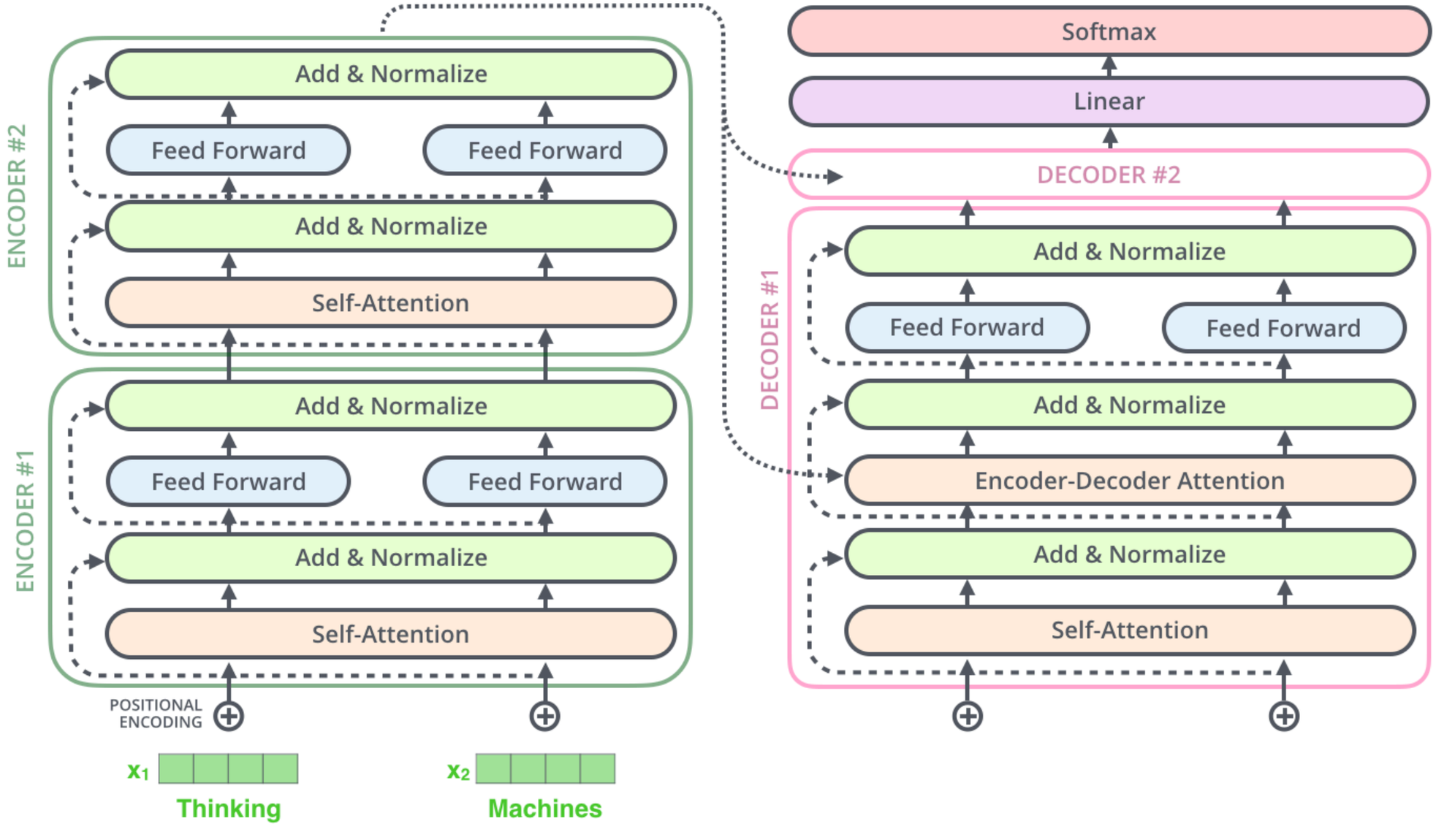


Figure from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

Transformer architecture detail; single encoder block

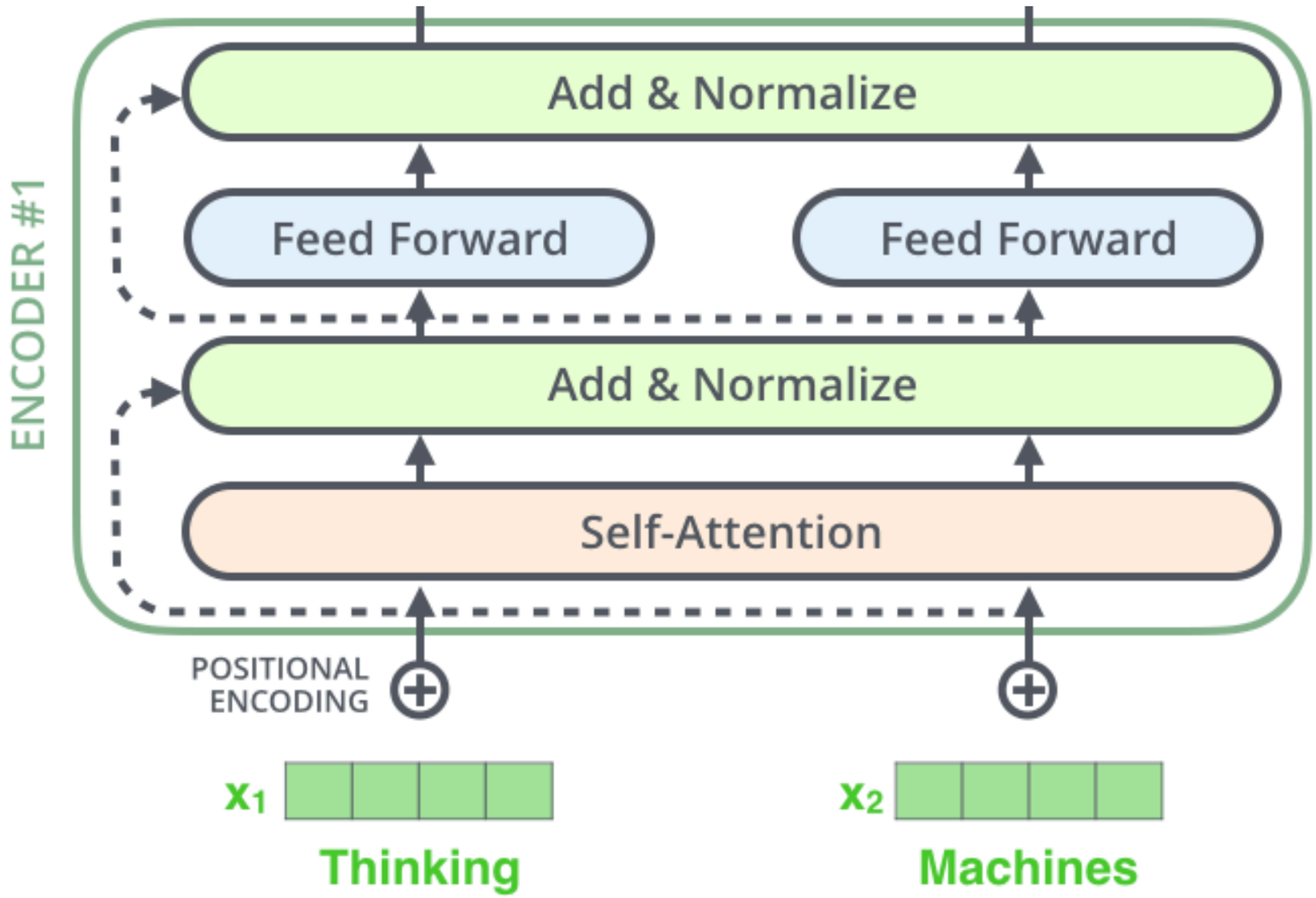


Figure adapted from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

As input, each hidden state initialized with representation of word **meaning** (context-free) and **position** in the sequence

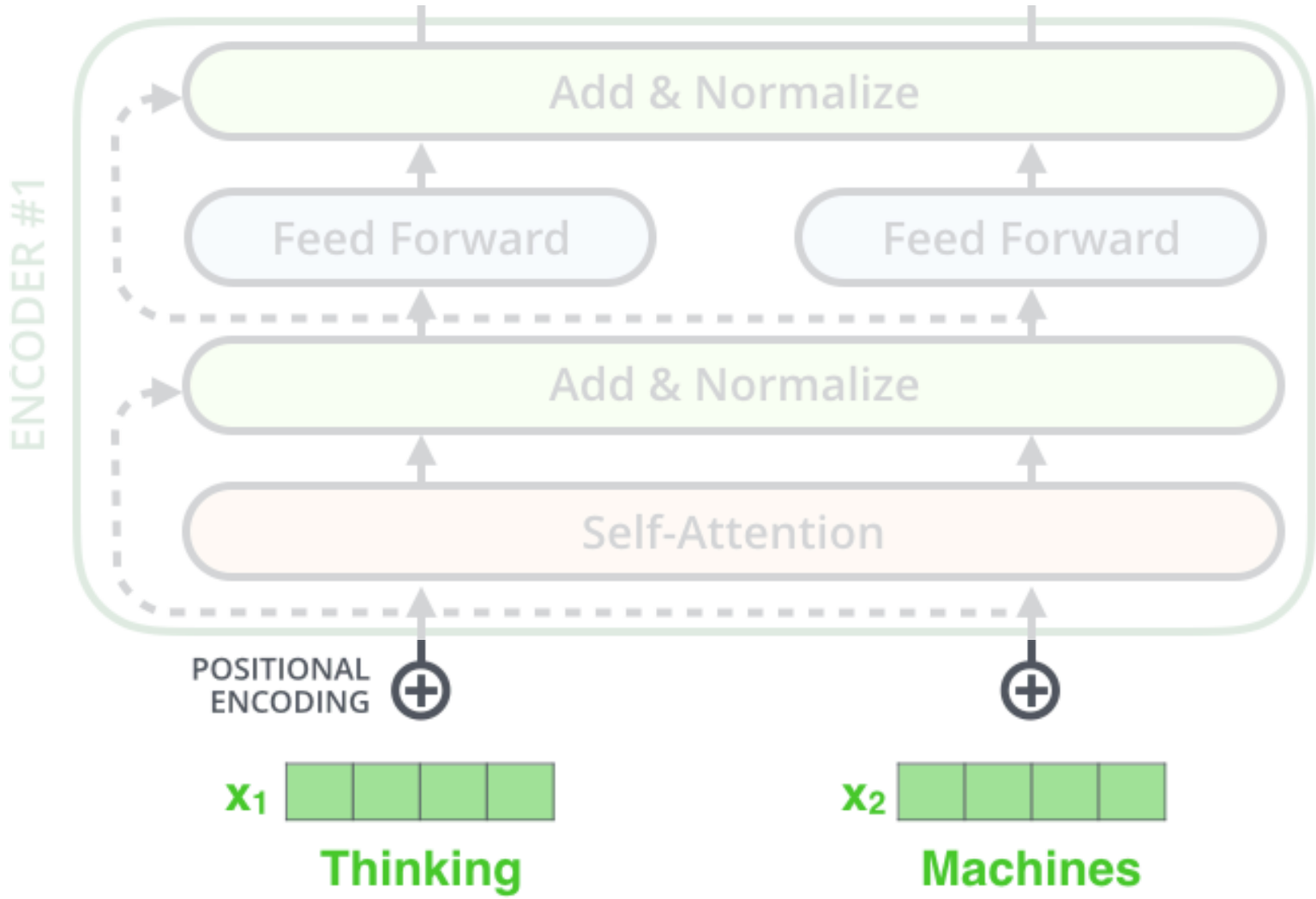


Figure adapted from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

Each block applies **self-attention** and a **feed-forward NN** (+normalization and residual connections)

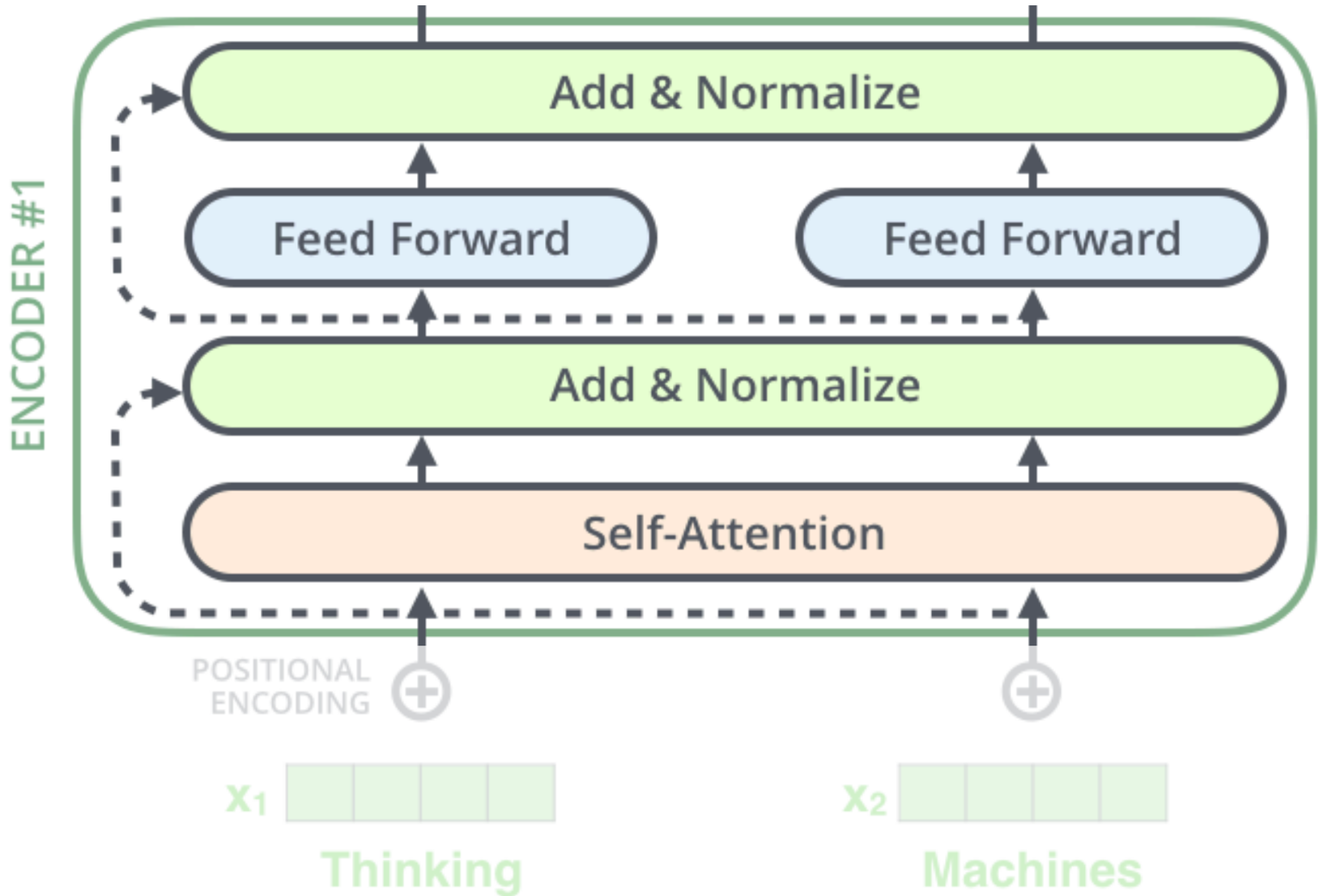


Figure adapted from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

Each block applies **self-attention** and a **feed-forward NN**
(+normalization and residual connections)

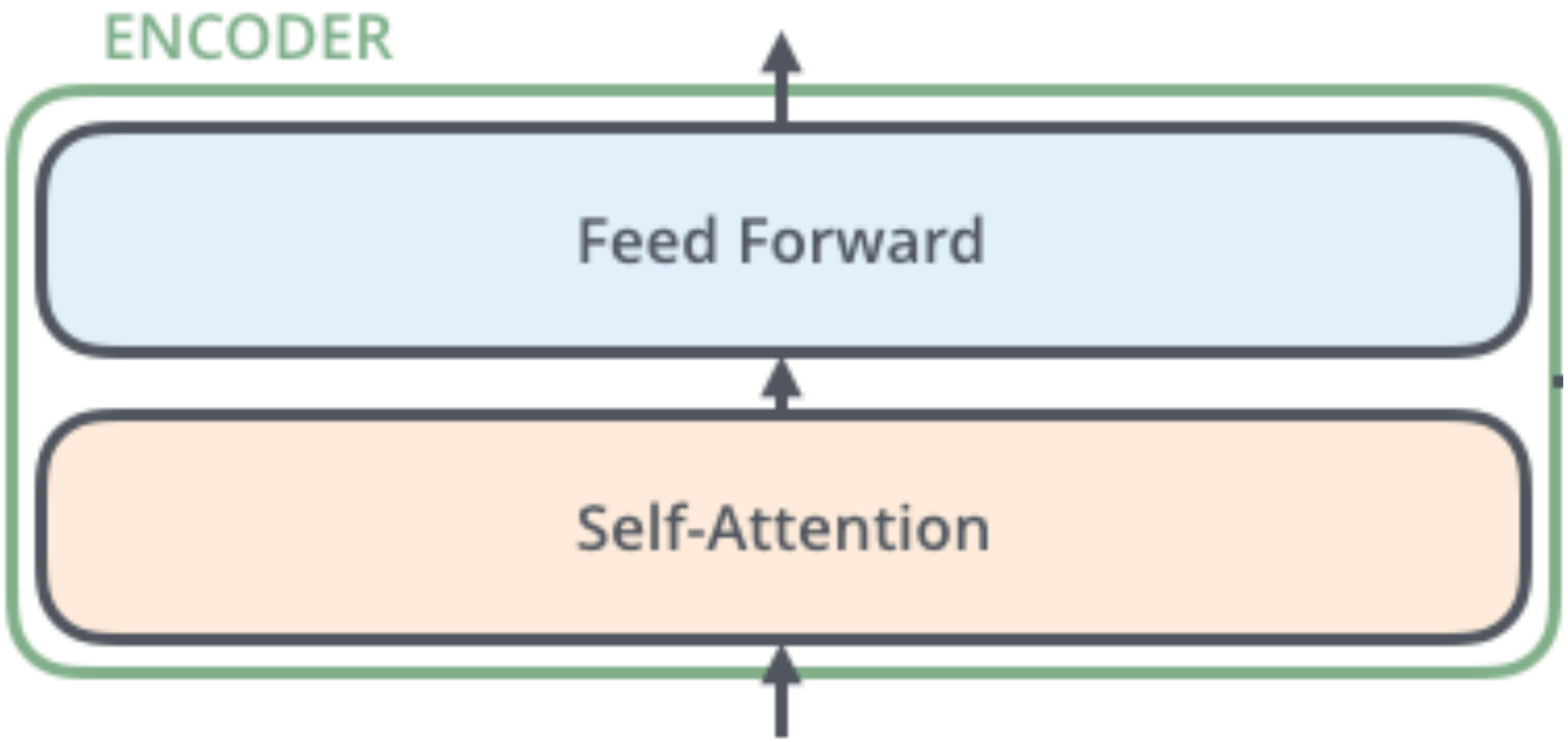
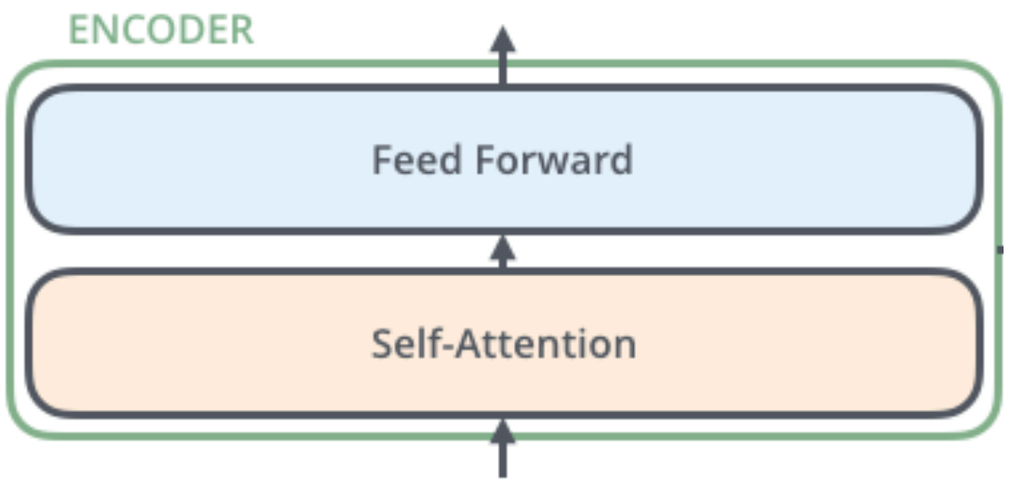


Figure adapted from http://jalammar.github.io/illustrated-transformer/

Transformer architecture

The steps performed in each block can be characterized as

1. **Communication** (attention): access information from context
2. **Computation** (feed-forward NN): update state based on info




```
class Block(nn.Module):
    def forward(self, x):
        x = x + self.attn(self.ln_1(x))
        x = x + self.mlpf(self.ln_2(x))
        return x
[…]
h = [Block() for _ in range(n_layer)]
```

Figure from http://jalammar.github.io/illustrated-transformer/, code adapted from https://github.com/karpathy/minGPT

Transformer architecture

The **feed-forward NNs** are standard single hidden layer fully connected networks with a non-linear activation (e.g. GeLU)

(Projection to 4x hidden size common, but other values also OK)

```
self.mlp = ModuleDict(dict(
  c_proj  = Linear(4 * config.n_embd, config.n_embd),
  act     = GELU(),
  c_fc    = Linear(config.n_embd, 4 * config.n_embd),
  dropout = Dropout(config.resid_pdrop),
))
m = self.mlp
self.mlpf = lambda x: m.dropout(m.c_proj(m.act(m.c_fc(x)))) # MLP forward
```

Code adapted from https://github.com/karpathy/minGPT

Transformer architecture

The **self-attention** mechanism involves computing **query**, **key** and **value** vectors for each hidden state using learned projections

The similarity (dot product) of queries and keys is then used to determine which values are used to update the hidden state

Information retrieval (web search) analogy:

- **Query**: word or phrase searched for
- **Key**: representation of document in search engine
- **Value**: the document itself

Transformer architecture

For a well-documented implementation of a basic GPT model, see e.g. https://github.com/karpathy/minGPT (~300 LOC in model.py)

```
class CausalSelfAttention(nn.Module):
[…]
    def forward(self, x):
        B, T, C = x.size() # batch size, sequence length, embedding dimensionality (n_embd)

        # calculate query, key, values for all heads in batch and move head forward to be the batch dim
        q, k ,v  = self.c_attn(x).split(self.n_embd, dim=2)
        k = k.view(B, T, self.n_head, C // self.n_head).transpose(1, 2) # (B, nh, T, hs)
        q = q.view(B, T, self.n_head, C // self.n_head).transpose(1, 2) # (B, nh, T, hs)
        v = v.view(B, T, self.n_head, C // self.n_head).transpose(1, 2) # (B, nh, T, hs)

        # causal self-attention; Self-attend: (B, nh, T, hs) x (B, nh, hs, T) -> (B, nh, T, T)
        att = (q @ k.transpose(-2, -1)) * (1.0 / math.sqrt(k.size(-1)))
        att = att.masked_fill(self.bias[:,:,:T,:T] == 0, float('-inf'))
        att = F.softmax(att, dim=-1)
        att = self.attn_dropout(att)
        y = att @ v # (B, nh, T, T) x (B, nh, T, hs) -> (B, nh, T, hs)
        y = y.transpose(1, 2).contiguous().view(B, T, C) # re-assemble all head outputs side by side

        # output projection
        y = self.resid_dropout(self.c_proj(y))
        return y
```

Transformer architecture

An attempt at a rough analogy for transformer operations:

Imagine a collaborative game where players are each locked in a room, given a word (e.g. "light") and its position in a sentence (e.g. 3rd word), and tasked to discover the contextual meaning of the word.

The only form of communication is by a mechanism where each player writes three brief notes: a question they want answered (query), what questions they can answer (key), and the information they want to communicate (value).

An outside party compares question and answer notes and provides each player is with the information note corresponding to the best match to their question.

Attention

Players then get to think about what they believe given this information.

MLP

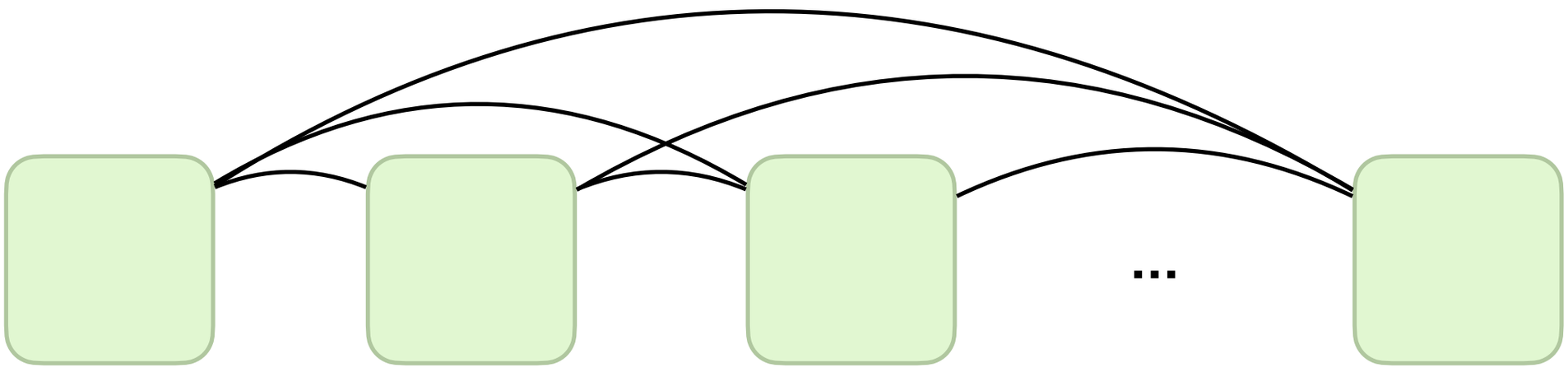
The above steps are repeated *n* times (layers), giving the players an opportunity to progressively refine their understanding.

Transformer architecture

Self-attention involves computing relevance of each word to each other word in context (*quadratic*!)

By contrast to LMs using convolutional (CNN) and recurrent (RNN) architectures, the computation required to pass information from one word representation to another is constant

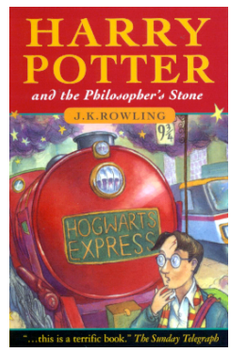
→ Transformers "remember" everything in the context



Transformer architecture

The attention mechanism is both one of the greatest strengths and a fundamental limitation of the transformer architecture

- Allows models to efficiently use considerably **longer contexts** (input/output text) than previous architectures, in many leading LLMs over **100,000 words** ≈



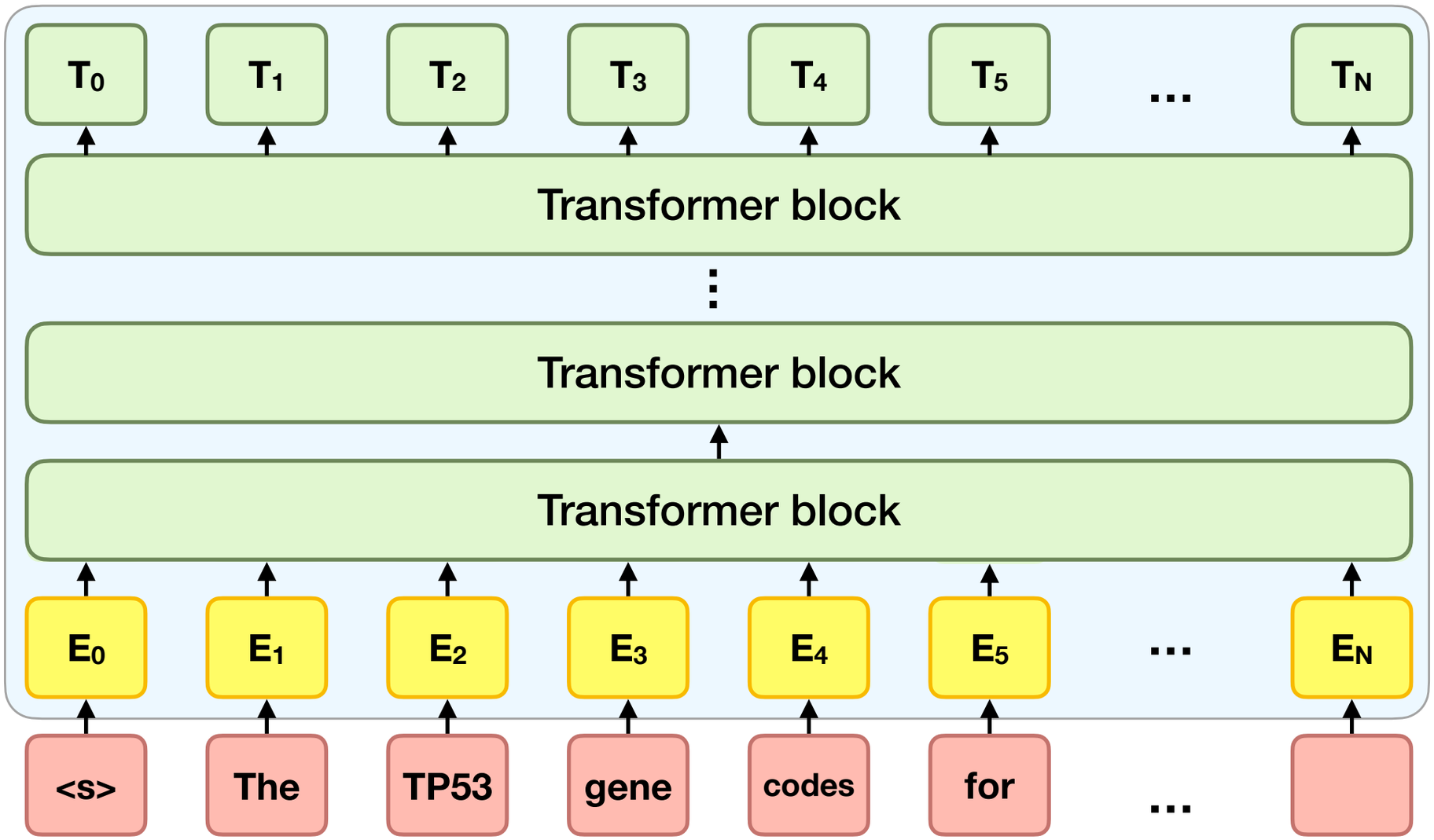
- The basic attention mechanism is **quadratic in space and computational requirements**, limiting the maximum context length

(Relieving this bottleneck is a major focus of research)

Inputs and outputs

Input words are mapped to embeddings that are progressively transformed to produce contextual representations

For task-specific fine-tuning, **output** layers typically added on top



Classes of transformer models

The original **encoder-decoder** transformer architecture can be applied to any **text-to-text** task, not just machine translation

The **encoder** and **decoder** components of the original transformer architecture can be used separately

Three broad classes of transformer models:

- **Encoder-only** (e.g. BERT): contextualized representations of input words
- **Decoder-only** (e.g. GPT): generates text conditioned on previous words → causal language model
- **Encoder-decoder** (e.g. T5): general text-to-text mapping

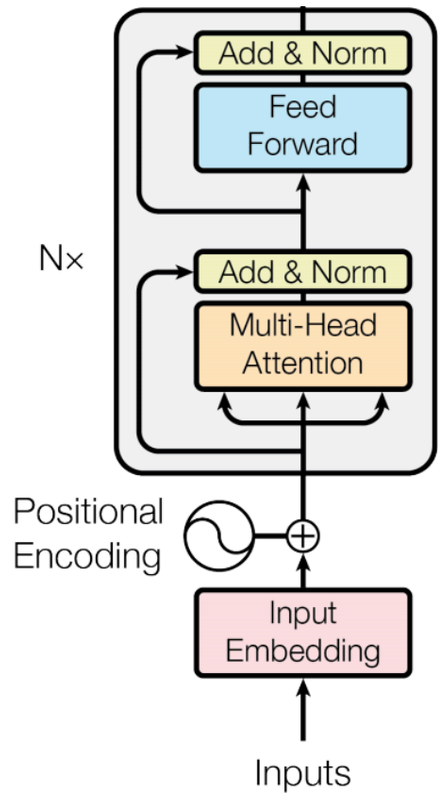
Bidirectional Encoder Representations from Transformers (BERT)

Encoder-only model introduced by Devlin et al. (Google) in 2019

English model pretrained on 3B words (Wikipedia + BooksCorpus)

Model sizes: 110M parameters (base) and 340M parameters (large)

Fine-tuning for various language understanding tasks showed results surpassing human performance



Devlin *et al*. (2019) *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*

Generative Pre-trained Transformer (GPT)

Decoder-only models first introduced by Radford et al. (OpenAI) in 2018

GPT-1 pre-trained on ~1B word BooksCorpus (English)

Fine-tuning for supervised NLP tasks, advanced SOTA

Followed up by **GPT-2**, **GPT-3**, **InstructGPT**, **ChatGPT**, and **GPT-4** and **GPT-5** models

Model sizes: 117M (GPT-1) to 175B (GPT-3); GPT-4→ unknown, GPT OSS 117B (5B active)

Radford *et al*. (2018) *Improving Language Understanding by Generative Pre-Training*

Generative Pre-trained Transformer (GPT)

Types of generative language models by training process

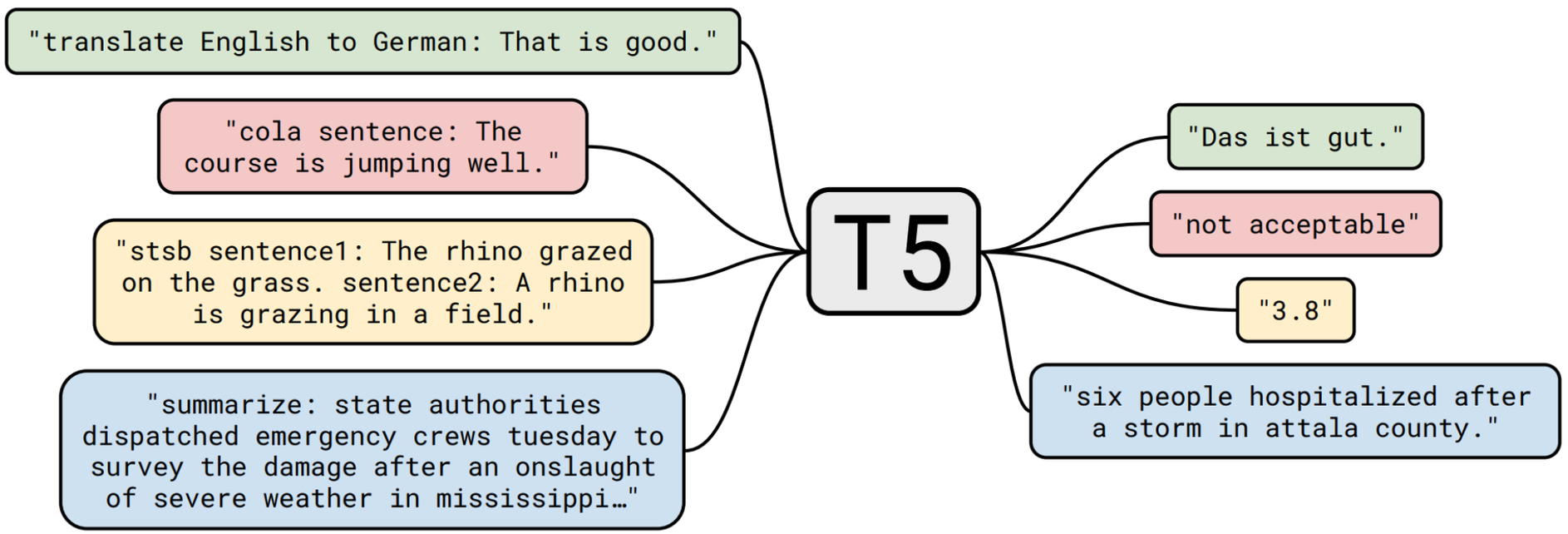
- **Base language models** (e.g. GPT1-3): trained on "naturally occurring" texts such as books and web pages — poor "UI"
- **Instruction-tuned models** (e.g. InstructGPT): trained on texts with an instruction - context - response structure
- **Chat models** (e.g. ChatGPT, GPT-4): trained on texts with explicit dialogue structure
- "**Thinking**" or "**reasoning**" **models** (e.g. GPT-4o, GPT-5): trained on texts with "chains of thought" preceding response

Text-to-Text Transfer Transformer (T5)

Encoder-decoder models introduced by Raffel et al. (2020)

Models ranging from **60M to 11B parameters** trained on 34B words

Key proposal: cast NLP tasks as text-to-text, train single model to perform all tasks



Raffel et al. (2020) “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”

Selecting a class of transformer

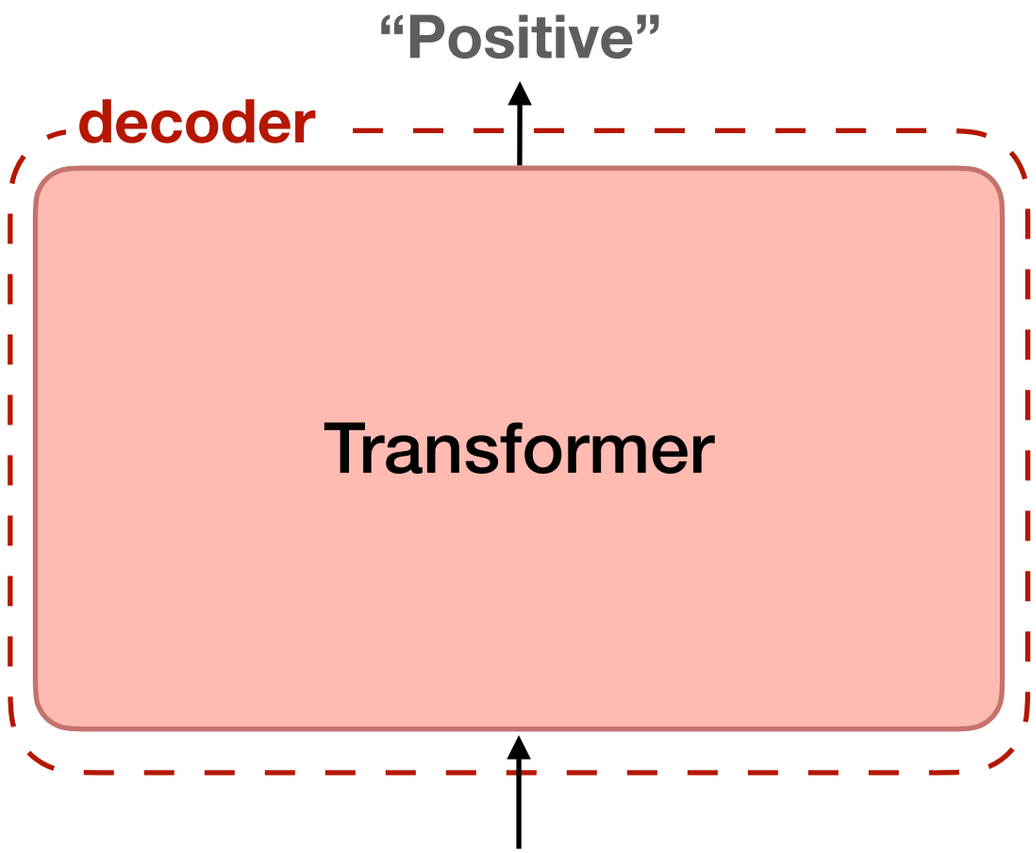
(Simplifying)

- **Encoder-only** ("BERT-like") models excel at **classification** tasks; larger models ~1B parameters (consumer hardware)
- **Decoder-only** ("GPT-like") models excel at **generation**; largest models 100s of billions of parameters (supercomputers)
- **Encoder-decoder** ("T5-like") models excel at **sequence-to-sequence** tasks; largest models 10s of billions of parameters

State-of-the-art encoder-only and encoder-decoder models often fine-tuned, the largest decoder-only models commonly used as-is in **zero- or few-shot settings**

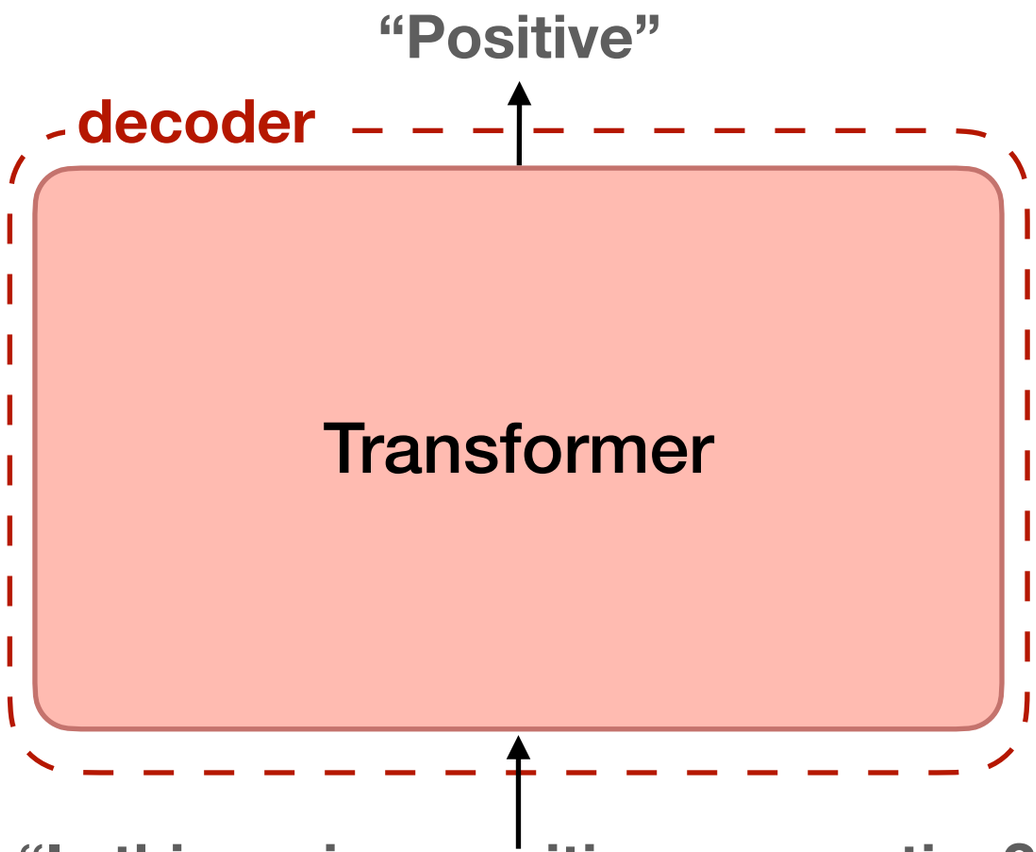
Zero- / few-shot prediction

Zero-shot classification



“Is this review positive or negative?
Review: This is a very good movie.
Answer: ”

One-shot classification



“Is this review positive or negative?
Review: This movie sucks!
Answer: Negative

Is this review positive or negative?
Review: This is a very good movie.
Answer: ”

https://tinyurl.com/unpack-pipeline

Text mining with transformers

Approaches

All classes of transformer models can be applied to information extraction and text mining, for example:

- **Encoder-only**: cast tasks as classification, fine-tune separate models for each task
- **Decoder-only**: develop natural language prompts for task, generate using pre-trained instruction / chat model
- **Encoder-decoder**: cast task as text-to-text with document as input and structured data as output, fine-tune single model

We'll here focus on the first in the context of a "traditional" information extraction pipeline

Text mining pipeline

Source texts (e.g. PubMed)

Document selection

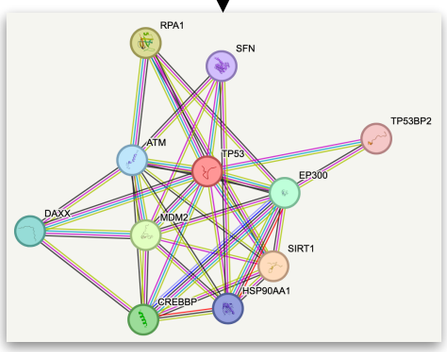
Named entity recognition

Normalization

Relation extraction

Structured data

Analysis



Useful information

Named entity recognition

Example: taxonomic name mention recognition

1	Primary structure of cytochrome c gene from the white root rot Kingdom fungus Species Rosellinia necatrix.
2	The nucleotide sequence of the cytochrome c (CytC) gene of the white root rot Kingdom fungus Species Rosellinia necatrix was analyzed.
3	The structure of this gene, which had three introns in the coding region, was similar to that of Species Aspergillus nidulans.
4	The second intron of the Species R. necatrix CytC gene was not present in Species Neurospora crassa or Species Fusarium oxysporum.
5	However, the amino acid sequence of Species R. necatrix was most similar to that of Species Neurospora crassa.
6	Thus, it seemed that the second intron of the Species R. necatrix CytC gene was inserted into its present position after Species R. necatrix and its closest relatives diverged evolutionarily.

Luoma et al. (2023) "S1000: A better taxonomic name corpus for biomedical information extraction"

Normalization

Example: taxonomic name normalization to NCBI Taxonomy

1 Primary structure of cytochrome c gene from the white root rot fungus Rosellinia necatrix.

2 The nucleotide sequence of the cytochrome c (CytC) gene of the white root rot fungus Rosellinia necatrix was analyzed.

3 The structure of this gene, which had three introns in the coding region, was similar to that of Aspergillus nidulans.

4 The second intron of the R. necatrix CytC gene was not present in Neurospora crassa or Fusarium oxysporum.

5 However, the amino acid sequence of R. necatrix was most similar to that of Neurospora crassa.

6 Thus, it seemed that the second intron of the R. necatrix CytC gene was inserted into its present position after R. necatrix and its closest relatives diverged evolutionarily.

Species "R. necatrix" ID:T4

Taxonomy:77044

Scientific name:Rosellinia necatrix

rank:species

Relation extraction

Example: `Complex formation` relation extraction

1 Rat protein tyrosine phosphatase eta physically interacts with the PDZ domains of syntenin.

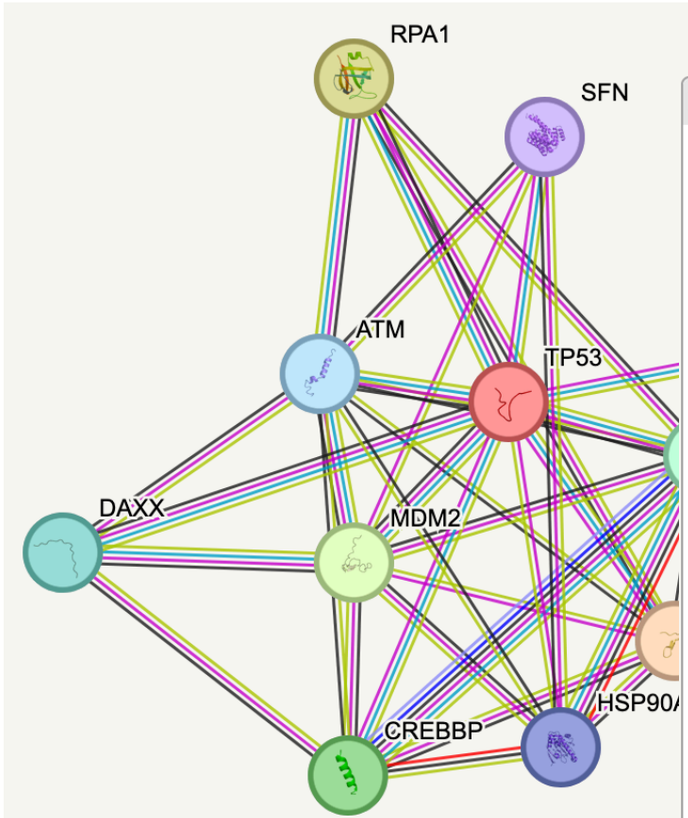
2 The tyrosine phosphatase r-PTPeta is able to suppress the malignant phenotype of rat thyroid tumorigenic cell lines.

3 To identify r-PTPeta interacting proteins, a yeast two-hybrid screening was performed and an insert corresponding to the full-length syntenin cDNA was isolated.

4 It encodes a protein containing two PDZ domains that mediates the binding of syntenin to proteins such as syndecan, proTGF-alpha, beta-ephrins and neurofascin.

Analysis

Example: STRING DB interaction network for human p53



Interaction

TP53 *[ENSP00000269305]*

Cellular tumor antigen p53; Acts as a tumor suppressor in many tumor types; induces growth arrest or apoptosis depending on the physiological circumstances and cell type. Involved in cell cycle regulation as a trans-activator that acts to negatively regulate cell division by controlling a set of genes required for this process. One of the activated genes is an inhibitor of cyclin-dependent kinases. Apoptosis induction seems to be mediated either by stimulation of BAX and FAS antigen expression, or by repression of Bcl-2 expression. Its pro-apoptotic activity is activated via its intera [...]

↔

MDM2 *[ENSP00000258149]*

E3 ubiquitin-protein ligase Mdm2; E3 ubiquitin-protein ligase that mediates ubiquitination of p53/TP53, leading to its degradation by the proteasome. Inhibits p53/TP53- and p73/TP73-mediated cell cycle arrest and apoptosis by binding its transcriptional activation domain. Also acts as a ubiquitin ligase E3 toward itself and ARRB1. Permits the nuclear export of p53/TP53. Promotes proteasome-dependent ubiquitin-independent degradation of retinoblastoma RB1 protein. Inhibits DAXX-mediated apoptosis by inducing its ubiquitination and degradation. Component of the TRIM28/KAP1-MDM2-p53/TP53 [...]

Evidence suggesting a functional link:

Neighborhood in the Genome:	none / insignificant.	
Gene Fusions:	none / insignificant	
Cooccurence Across Genomes:	none / insignificant	
Co-Expression:	yes (score 0.049). In addition, putative homologs are coexpressed in other organisms (score 0.109).	show
Experimental/Biochemical Data:	yes (score 0.999). In addition, putative homologs were found interacting in other organisms (score 0.185).	show
Association in Curated Databases:	yes (score 0.900).	show
Co-Mentioned in Pubmed Abstracts:	yes (score 0.999). In addition, putative homologs are mentioned together in other organisms (score 0.100).	show
Combined Score:	0.999	

Text vs. structured data

Input: "Tyrosine phosphatase eta physically interacts with the PDZ domains of syntenin."

Output (names + normalization):

ID	Type	Start	End	Text	DBRef
T0	Protein	0	24	Tyrosine phosphatase eta	Gene:24326
T1	Protein	70	78	Syntenin	Gene:83841

Output (relations):

ID	Type	Arg1	Arg2
T0	Complex formation	T0	T1

Approach

Many information extraction tasks can be straightforwardly cast as supervised classification, e.g.:

- **Document selection** → text (word sequence) classification
- **Named entity recognition** → token (word) classification
- **Relation extraction** → text classification

We'll briefly look at these three next

(Normalization can be cast e.g. as ranking pairs of text, one representing mention in context and the other a candidate entity definition)

Document selection

Document selection is a simple text classification task

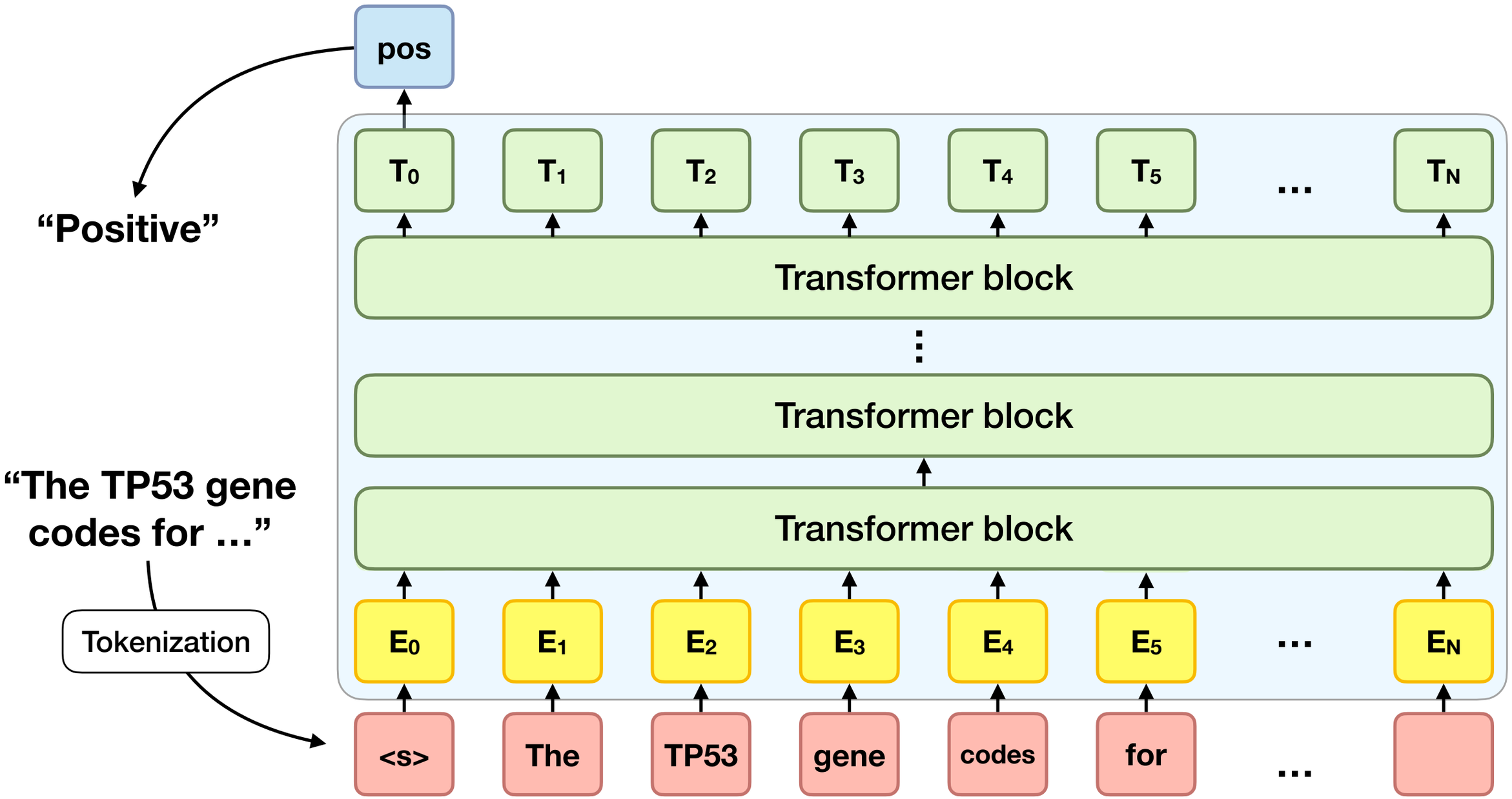
Input: text, represented as sequence of words

Output: label (or set of labels) from predefined categories (e.g. `Positive`/`Negative`, `Relevant`/`Irrelevant`)

Approach: attach output layer e.g. to start-of-text token or a pooling layer (e.g. max-pool)

(multiclass vs. multilabel classification is mostly a matter of choosing appropriate activation and loss functions)

Text classification



Named entity recognition

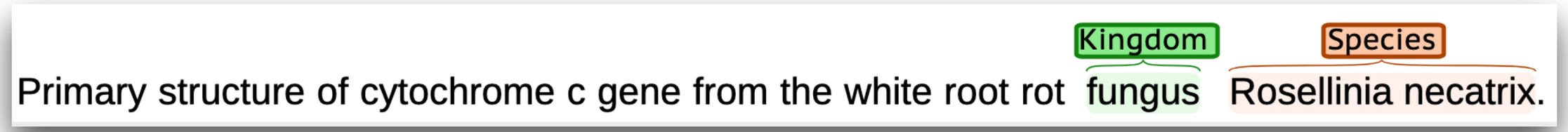
The recognition of (non-overlapping) mentions of names of entities of interest can be cast as token classification:

Input: text, represented as sequence of words

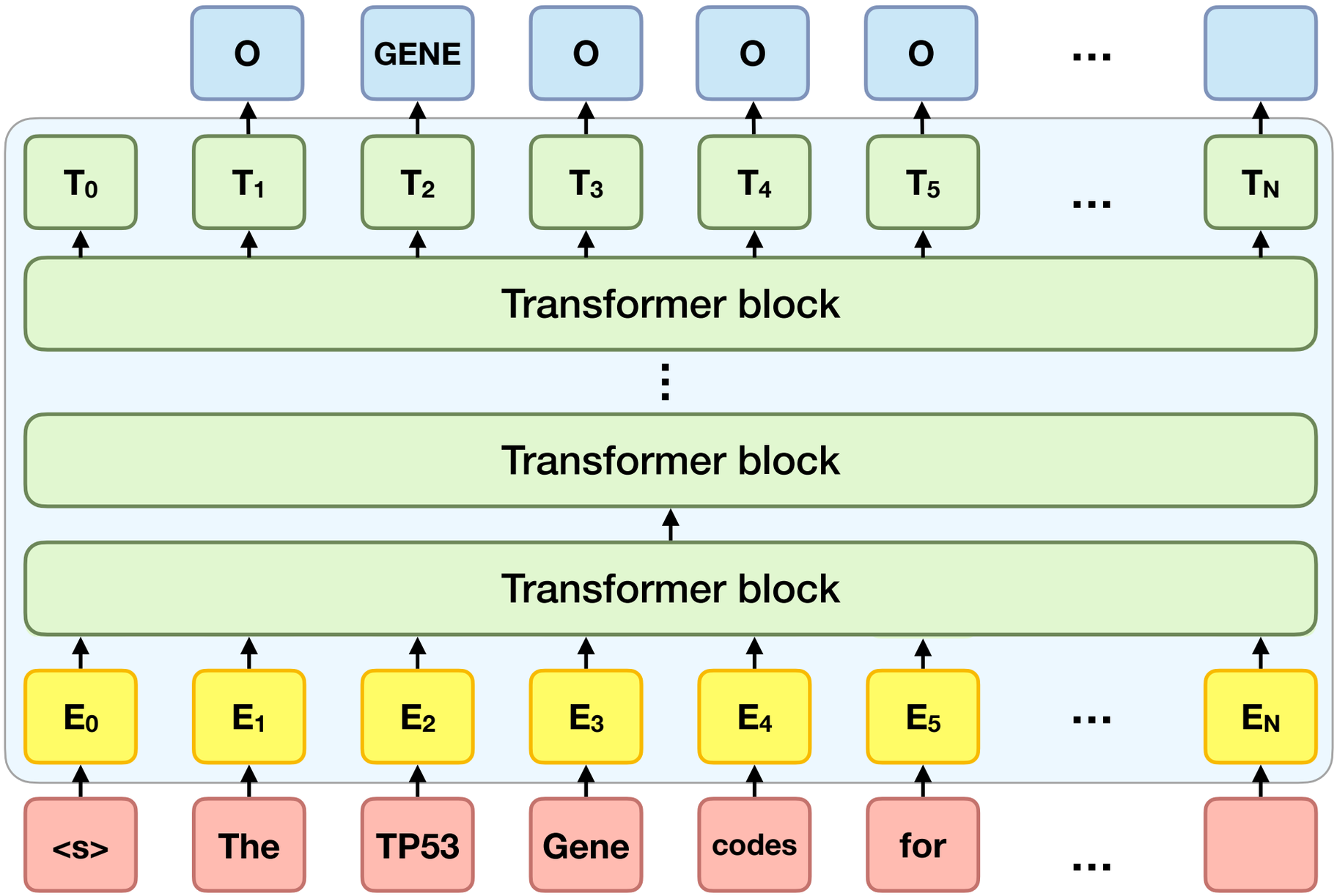
Output: sequence of labels (one per word) from predefined categories (e.g. `Gene`, `Disease`, `Species`, etc.)

(glossing over boundary issues and IOB encoding)

Approach: attach output layer to each token, fine-tune with gold standard data



Token (word) classification



Relation extraction

Identification and classification of all (typically binary) relations of interest between mentions of entities of interest (from NER)

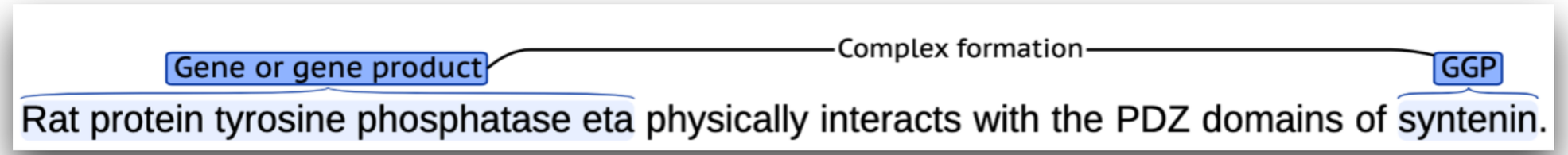
In a pipeline, for example:

Input: text (sequence of words) and entity mentions (types and offsets)

Output: triples of (start-entity, end-entity, relation-type)

Approach: consider all pairs of entity mentions, mark each one in turn in text, cast as text classification with relation type as label

e.g. "<E1>tyrosine phosphatase eta</E1> physically interacts with the PDZ domains of <E2>syntenin</E2>." → `Complex formation`

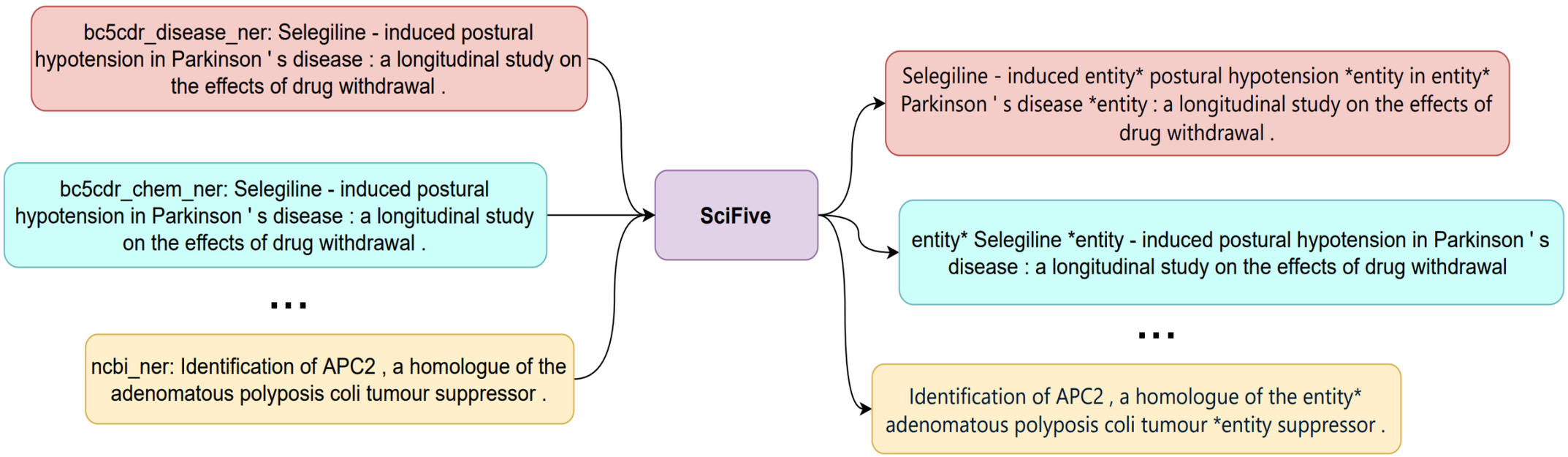


End-to-end approach

Instead of breaking down the mapping from input text to structured representation into subtasks, IE could be performed using and **end-to-end** approach using e.g. a text-to-text model:

Input: text (e.g. PubMed abstract)

Output: structured representation



Phan et al. (2021) "SciFive: a text-to-text transformer model for biomedical literature"

Just ask GPT?

Large causal language models are capable of *attempting* effectively any text mining task out of the box

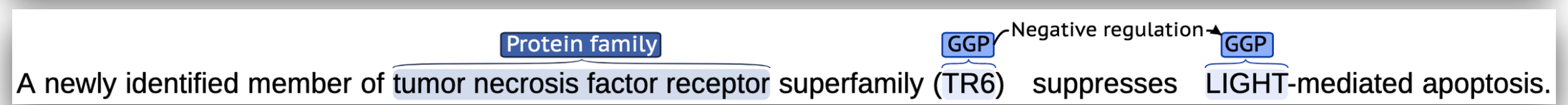
→ Simply formulate an appropriate prompt

Issues: lack of fine-grained **control**, **cost** of running massive models at very large scale

List the protein names appearing in the following text:

A newly identified member of tumor necrosis factor receptor superfamily (TR6) suppresses LIGHT-mediated apoptosis.

1. Tumor necrosis factor receptor superfamily (TR6)
2. LIGHT



Experimental setups

Data for NLP tasks is often prepared with conventional supervised ML setups, split into

Training: for training (or fine-tuning) models

Validation: for evaluating models during development (e.g. hyperparameter optimization)

Test: for evaluating final model

```
from datasets import load_dataset

d = load_dataset('spyysalo/bc2gm_corpus')
print(d)
```

```
DatasetDict({
    train: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 12500
    })
    validation: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 2500
    })
    test: Dataset({
        features: ['id', 'tokens', 'ner_tags'],
        num_rows: 5000
    })
})
```

Experimental setups

LLM evaluation datasets are often used in zero- or few-shot settings **without changes to the pre-trained model** and may not contain training data

```python
from datasets import load_dataset

d = load_dataset('cais/mmlu', 'high_school_biology')
print(d)
```

```
DatasetDict({
    test: Dataset({
        features: ['question', 'subject', 'choices', 'answer'],
        num_rows: 310
    })
    validation: Dataset({
        features: ['question', 'subject', 'choices', 'answer'],
        num_rows: 32
    })
    dev: Dataset({
        features: ['question', 'subject', 'choices', 'answer'],
        num_rows: 5
    })
})
```

Experimental setups

Regardless of model category, standard rules for good experimental practice apply: do not train on data intended for evaluation, hold out test data throughout development, etc.

Closed models pretrained on datasets crawled from the internet can make this impossible: without information on the training data we cannot be sure that some test data has not been included ("contamination")

(For a light take on a serious issue, see Schaeffer 2023 *Pretraining on the Test Set Is All You Need*)

In practice

Practical exercises

Transformers have excellent support for all major deep learning libraries

We'll here focus on **PyTorch**, the library of choice for most work in the area

We'll be using the **Transformers** library and **Hugging Face Hub** resources

Code will be run on **Colab**

https://tinyurl.com/tmat-2025

A few words on resources first

Hugging Face 🤗

In recent years, Hugging Face (https://huggingface.co/) has emerged as a leading platform for Transformer models and tools

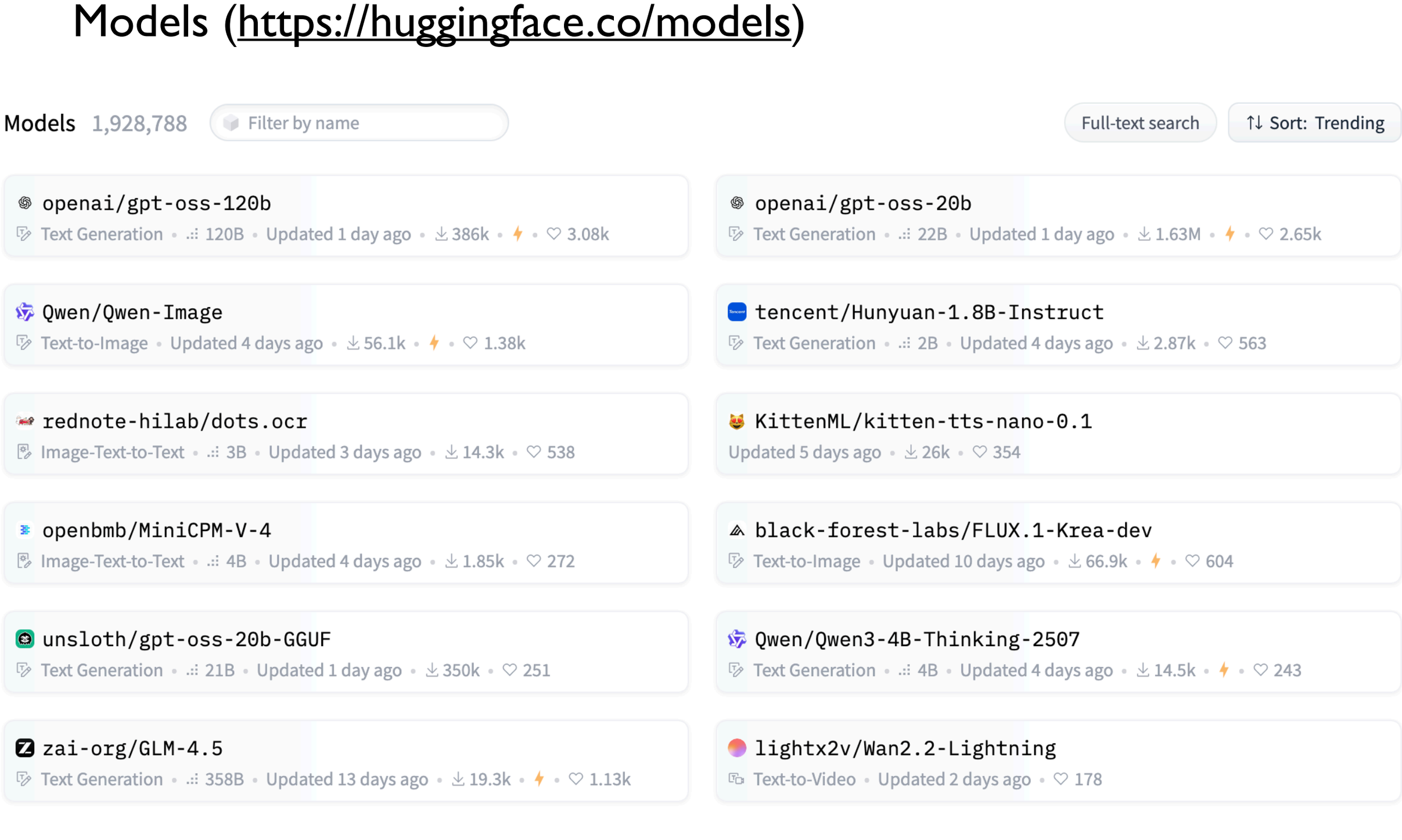
The Transformers Python library has extensive support for loading, training, and using transformer-based models

The Hugging Face Hub is a large repository including

- Models: ~2M models including both major pre-trained models as well as fine-tuned models for a broad range of tasks
- Datasets: ~500,000 datasets representing various tasks in NLP and other areas (e.g. sound and image processing)

Hugging Face 🤗

Models (https://huggingface.co/models)



Selected models

Hundreds of pre-trained "foundation" models for various languages and domains and hundreds of thousands of fine-tuned models are readily available

Selecting one that fits your needs can be challenging

We've already discussed **GPT**, **BERT**, and **T5**

We'll next look at a few with particular relevance to **biomedical text mining**

BioBERT

Encoder-only models trained by Lee et al. (2020) on **PubMed abstracts** and **PMC open access full text** publications

BERT base (**110M parameters**) and large (**340M parameters**) size models

Highly competitive but far from the only choice of a biomedical domain encoder-only model, also e.g. SciBERT, BlueBERT, PubMedBERT, BioMegatron, …

https://huggingface.co/dmis-lab/biobert-base-cased-v1.2

https://huggingface.co/dmis-lab/biobert-large-cased-v1.1

Lee et al. (2020) “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”

GALACTICA

Decoder-only models pre-trained on **106 billion words** from scientific sources by Meta

Model sizes range from **125M** (“mini”) to **120B** (“huge”) parameters

When published, the models established a new state-of-the-art in various tasks, including medical question answering

https://huggingface.co/facebook/galactica-1.3b

https://galactica.org/

Taylor et al. (2022) "Galactica: A Large Language Model for Science"

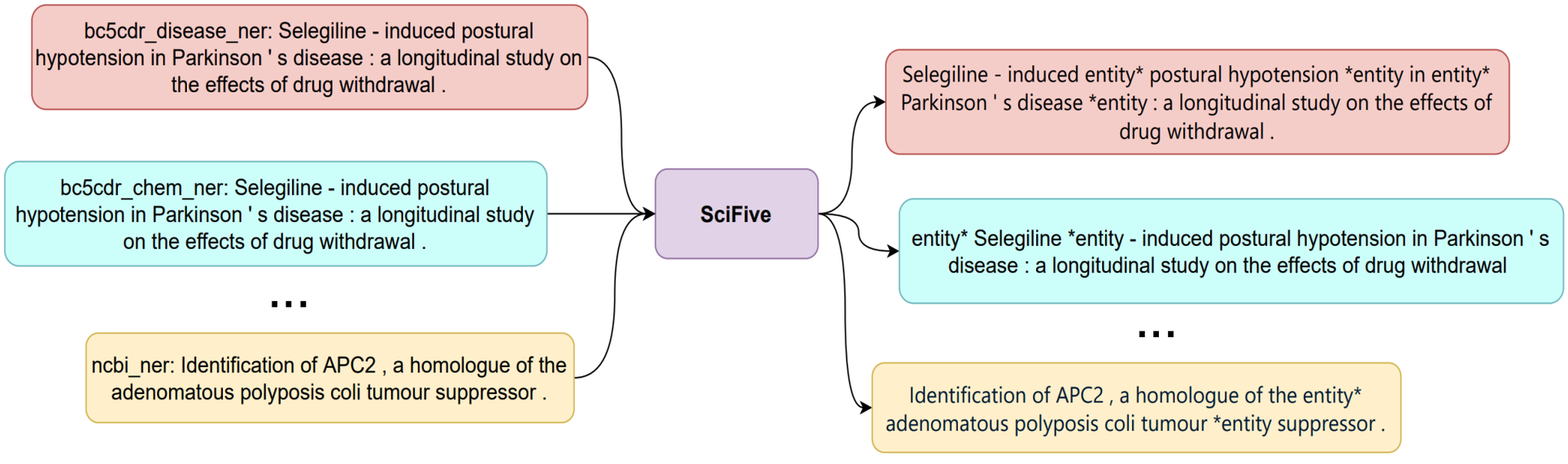
SciFive

Encoder-decoder models trained by Phan et al. (2021)

Model sizes T5-base (**220M parameters**) and T5-large (**770M parameters**)

https://huggingface.co/razent/SciFive-base-Pubmed_PMC

https://huggingface.co/razent/SciFive-large-Pubmed_PMC



Phan et al. (2021) "SciFive: a text-to-text transformer model for biomedical literature"

Biomedical datasets

There is no central repository of datasets for biomedical text mining

The **BigScience Biomedical Datasets** collaborative effort has compiled a set of over 100 biomedical domain datasets for e.g.

- Gene/protein, chemical, disease and organism name NER
- Normalization for various biomedical entities
- Protein-protein binding and chemical-protein relations
- Event extraction for various biomedical processes

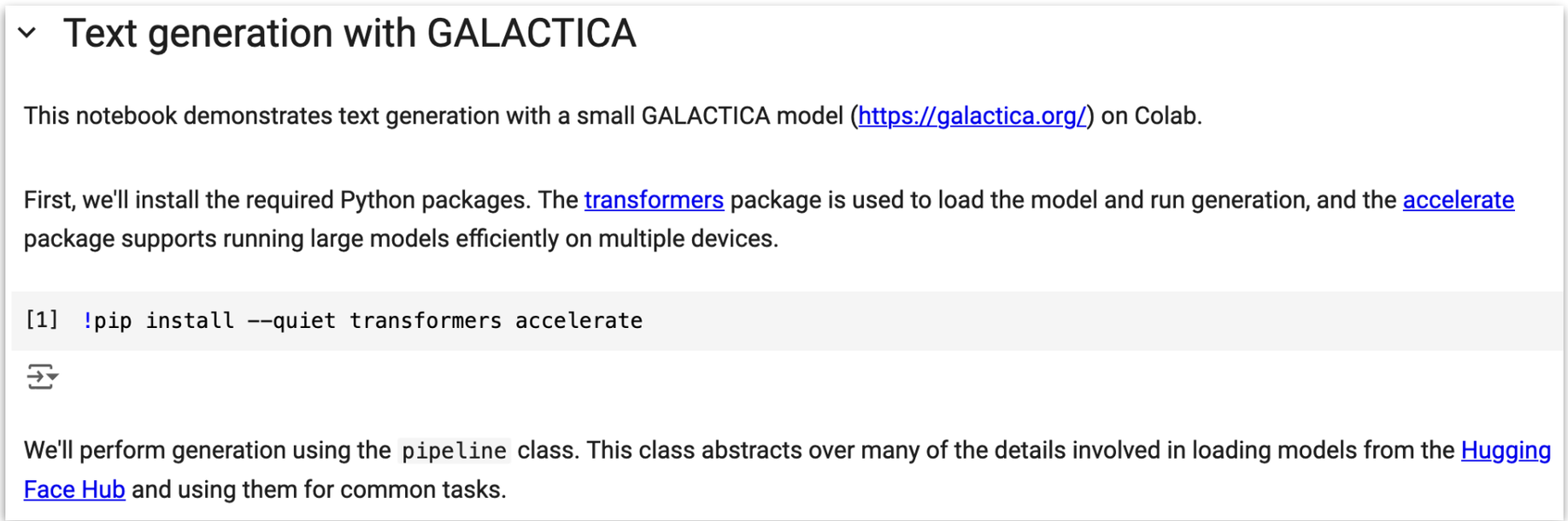
https://huggingface.co/bigbio

Text generation

GALACTICA text generation notebook

Text generation using a causal LM and LMs as knowledge base / article writing support

Try: prompt the model to generate some claim(s) relevant to your research. Are these correct?



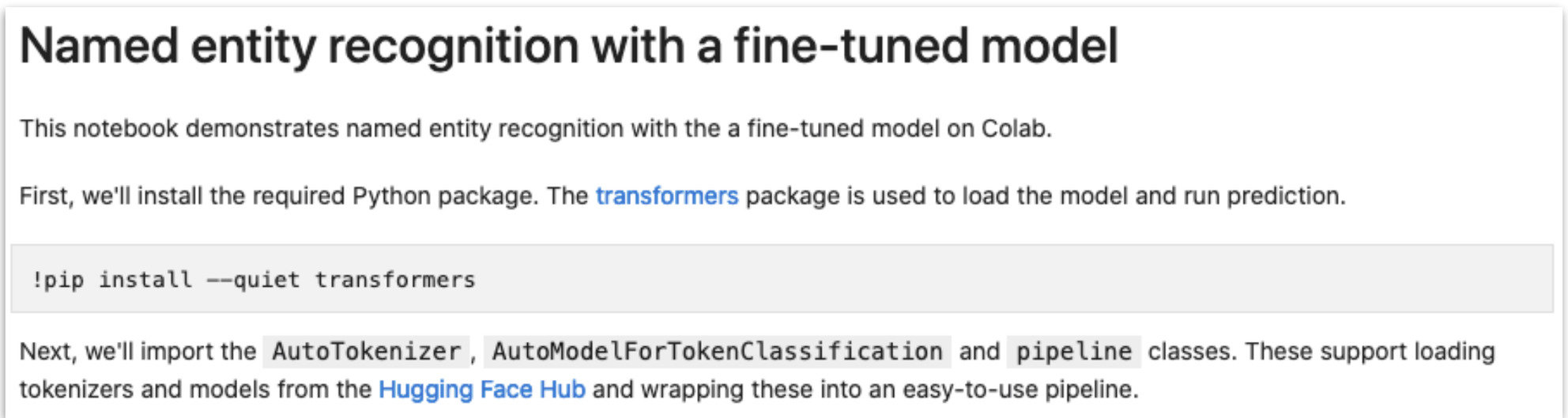
Taylor et al. (2022) "Galactica: A Large Language Model for Science"

Named entity recognition

BioBERT chemical NER notebook

NER using a encoder-only model fine-tuned on a chemical NER dataset

Try: provide the model with sentences that include mentions of chemicals and other entities (e.g. proteins). Does the model tag the correct mentions?



Named entity recognition with a fine-tuned model

This notebook demonstrates named entity recognition with the a fine-tuned model on Colab.

First, we'll install the required Python package. The transformers package is used to load the model and run prediction.

```
!pip install --quiet transformers
```

Next, we'll import the `AutoTokenizer`, `AutoModelForTokenClassification` and `pipeline` classes. These support loading tokenizers and models from the Hugging Face Hub and wrapping these into an easy-to-use pipeline.

Fine-tuning for NER

NER fine-tuning notebook

NER using the original BERT model and a "general domain" dataset

Try: replace the model and dataset with biomedical domain ones, (e.g. BioBERT and `bc2gm_corpus`) to fine-tune a model for biomedical domain NER

Fine-tuning a model for NER

Let's train a transformer model on a Named Entity Recognition (NER) dataset.

Setup

Install the required Python packages:

```
!pip install --quiet transformers datasets evaluate seqeval accelerate
```