

# Language Model Programming: Lecture 1

Kyle Richardson<sup>α</sup>, Gijs Wijnholds<sup>β</sup>

Allen Institute for AI (AI2)<sup>α</sup>  
Leiden Institute of Advanced Computer Science (LIACS)<sup>β</sup>

July 2025



Leiden Institute of  
Advanced Computer  
Science

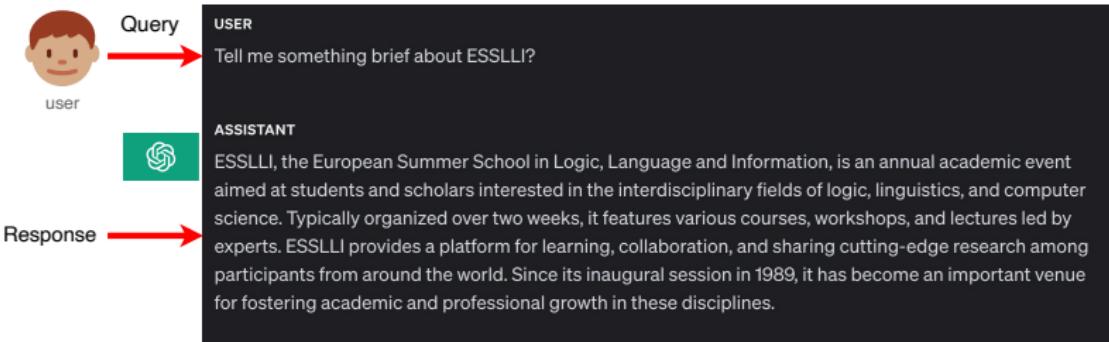


Universiteit  
Leiden

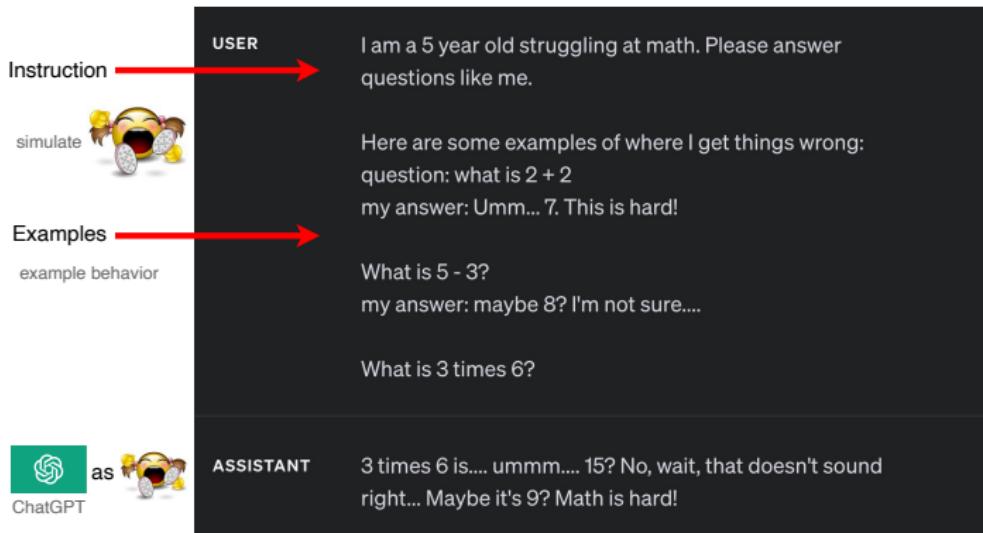
# Course Overview

# Language Model Programming

# Language models today



# In-context learning: learning through examples

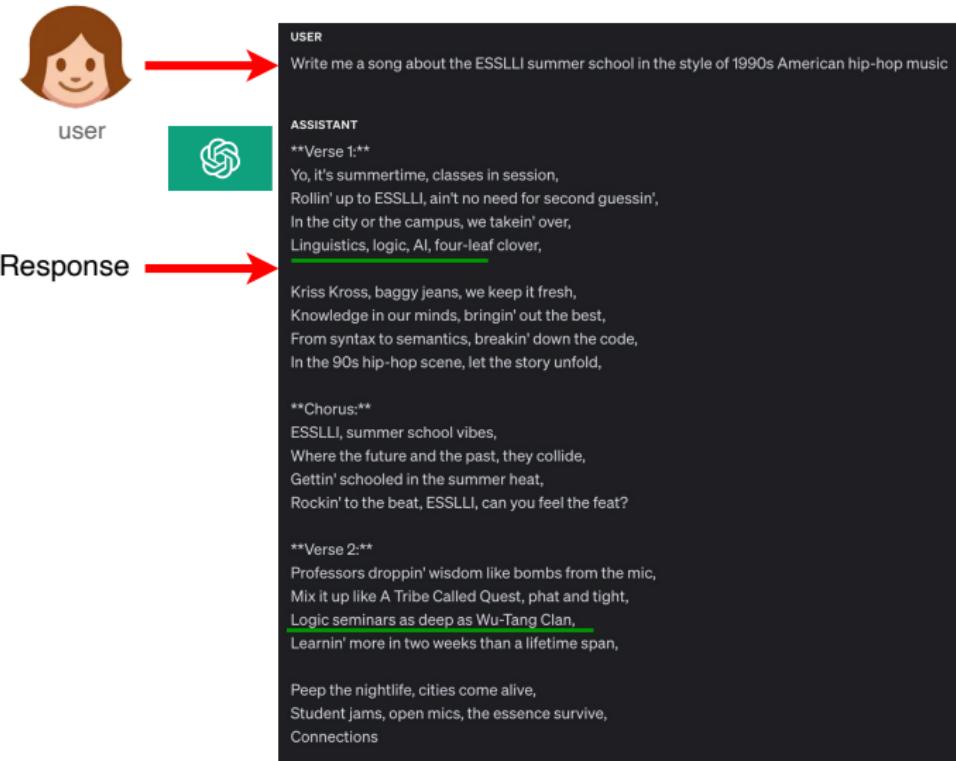


# In-context learning: learning through examples

<p>Instruction </p> <p>Examples example behavior   as </p>	<p>USER</p> <p>I am a 5 year old struggling at math. Please answer questions like me.</p> <p>simulate</p> <p>Here are some examples of where I get things wrong: question: what is <math>2 + 2</math> my answer: Umm... 7. This is hard!</p> <p>What is <math>5 - 3</math>? my answer: maybe 8? I'm not sure....</p> <p>What is 3 times 6?</p> <p>ASSISTANT</p> <p>3 times 6 is.... ummm.... 15? No, wait, that doesn't sound right... Maybe it's 9? Math is hard!</p>
---	--

**Traditional learning** coupled with **in-context learning** (no parameter updates, just new examples added to input)

# Abilities far exceed expectations...



# Abilities far exceed expectations...



USER

Write me a short function in Python that decides if a number is prime.



ASSISTANT

Certainly! Here's a short Python function that checks if a number is prime:

```
```python
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

```

Executable code



```
# Example usage:
print(is_prime(11)) # Output: True
print(is_prime(15)) # Output: False
...
```

```

# Integrating language models into complex systems

The screenshot shows the "ML Experiment Execution Engine" interface. It displays two steps of a "Machine learning experiment".

**Step 1:**

- Query:** Implement an encoder-only model using transformers that can do multiple-choice QA.
- System thought:** To implement an encoder-only model for multiple-choice question answering (QA) using the transformers library from Hugging Face, I'll first outline the steps in Python. This will involve loading a pre-trained encoder model, perhaps something like BERT or RoBERTa, and then adapting it to handle multiple-choice QA tasks. The model will receive a question and several possible answers as input and will have to select the most likely answer. I will use the `transformers` library for this purpose.
- Action (execute):**

```
from transformers import AutoModelForMultipleChoice, AutoTokenizer\n\nmodel_name = 'bert-base-uncased'\n\n# Load model and tokenizer\ntokenizer = AutoTokenizer.from_pretrained(model_name)\nmodel = AutoModelForMultipleChoice.from_pretrained(model_name)\n\nprint("Model and tokenizer loaded successfully.")
```
- Observation:**
  - Details:** Model and tokenizer loaded successfully.

**Step 2:**

- Running step 2**
- System thought:** The model and tokenizer have been successfully loaded. Next, I need to write a function that takes a question and a list of possible answers, formats them correctly for the model input, and then uses the model to predict the most likely answer. This function will use the tokenizer to encode the inputs and the model to get logits for each possible answer. Finally, it will select the answer with the highest logit as the predicted answer.
- Action (execute):**

```
def predict_answer(question, choices):\n    inputs = tokenizer([question] * len(choices), choices, padding=True, return\n        outputs = model(**inputs)
```

# Integrating language models into complex systems

The screenshot shows the ML Experiment Execution Engine interface. On the left, there's a sidebar with icons for a user profile, 'Machine learning experiment', and 'ChatGPT'. Below these are sections for 'Model generated code' and 'Observation'.

**Step 1:** A 'Running step 1' status bar at the top. The main area contains a 'Query' input field with the text 'Implement an encoder-only model using transformers that can do multiple-choice QA'. Below it is a 'System thought' section:

```
system thought : To implement an encoder-only model for multiple-choice question answering (QA) using the transformers library from Hugging Face, I'll first outline the steps in Python. This will involve loading a pre-trained encoder model, perhaps something like BERT or RoBERTa, and then adapting it to handle multiple-choice QA tasks. The model will receive a question and several possible answers as input and will have to select the most likely answer. I will use the transformers library for this purpose.
```

Below that is an 'Action (execute)' section with the following Python code:

```
from transformers import AutoModelForMultipleChoice, AutoTokenizer

model_name = 'bert-base-uncased'

# Load model and tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForMultipleChoice.from_pretrained(model_name)

print("Model and tokenizer loaded successfully.")
```

Under 'Observation', there's a 'Details' section showing the output: 'Model and tokenizer loaded successfully.'

**Step 2:** A 'Running step 2' status bar at the top. The main area contains a 'System thought' section:

```
system thought : The model and tokenizer have been successfully loaded. Next, I need to write a function that takes a question and a list of possible answers, formats them correctly for the model input, and then uses the model to predict the most likely answer. This function will use the tokenizer to encode the inputs and the model to get logits for each possible answer. Finally, it will select the answer with the highest logit as the predicted answer.
```

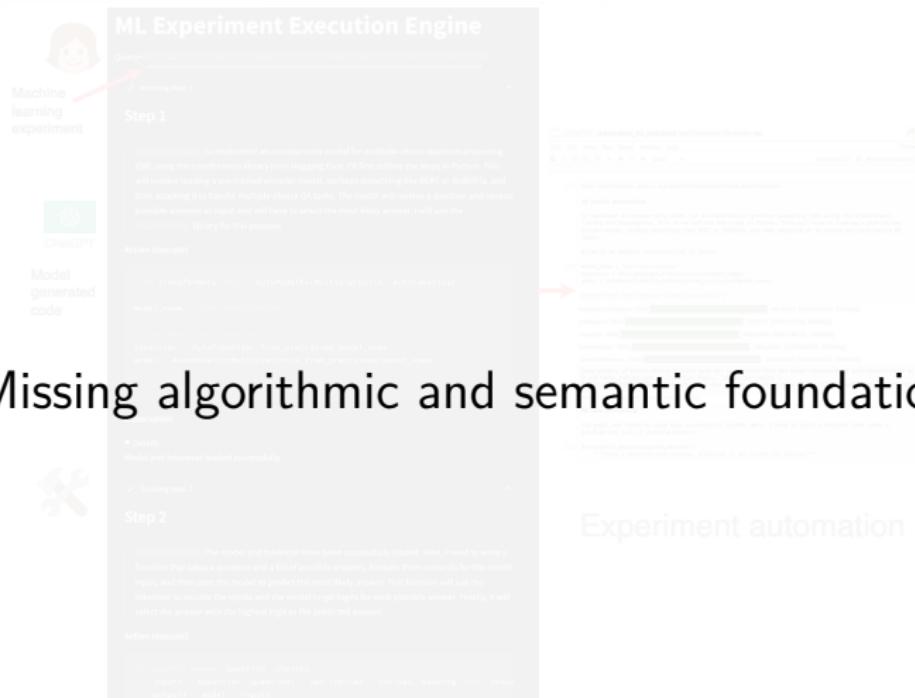
Below that is an 'Action (execute)' section with the following Python code:

```
def predict_answer(question, choices):
    inputs = tokenizer([question] * len(choices), choices, padding=True, return_outputs = model(**inputs)
```

On the right side of the interface, there's a separate window titled 'jupyter automated\_ml\_notebook.LoremCheckpoint.23 minutes ago' showing a Jupyter notebook cell with code and its execution results.

## Experiment automation

# Integrating language models into complex systems



# Integrating language models into complex systems

The screenshot shows the "ML Experiment Execution Engine" interface with two main sections: Step 1 and Step 2.

**Step 1:**

- Query:** "I want to implement an encoder-only model using transformers that can do multiple-choice QA."
- Action (execute):**

```
from transformers import AutoModelForMultipleChoice, AutoTokenizer\n\nmodel_name = 'bert-base-uncased'\n\n# Load model and tokenizer\ntokenizer = AutoTokenizer.from_pretrained(model_name)\nmodel = AutoModelForMultipleChoice.from_pretrained(model_name)\n\nprint("Model and tokenizer loaded successfully!")
```
- Observation:**
  - Running step 1
  - Model and tokenizer loaded successfully.

**Step 2:**

- System thought:** To implement an encoder-only model for multiple-choice question answering (MCQA) using the transformers library from Hugging Face, I'll first outline the steps in Python. This will involve loading a pre-trained encoder model, perhaps something like BERT or RoBERTa, and then adapting it to handle multiple-choice QA tasks. The model will receive a question and several possible answers as input and will have to select the most likely answer. I will use the transformers library for this purpose.
- Action (execute):**

```
def predict_answer(question, choices):\n    inputs = tokenizer(question + '\n' + '\n'.join(choices),\n                     padding=True,\n                     return_tensors='pt')\n\n    outputs = model(**inputs)
```
- Observation:**
  - Running step 2

A red arrow points from the "Observation" section of Step 1 to the "Action (execute)" section of Step 2, indicating the flow of the experiment.

In the background, a Jupyter notebook titled "automated\_ml\_notebook" is visible, showing the execution of the same code. A red arrow also points from the "Observation" section of Step 1 to the "Details" section of the Jupyter notebook, highlighting the connection between the two environments.

## Experiment automation

- Looks a lot like conventional programming, can we make the connection more precise?

Integrating language models into complex systems

## Experiment automation

- ▶ **Model programming:** How modeling components are assembled, how they interact, and the language of how this is specified.

# Problems are still easy to find



USER

Name me a country that is physically located  
between the Czech Republic and France



ASSISTANT

Germany is a country that is physically located  
between the Czech Republic and France.

# Problems are still easy to find



USER

Name me a country that is physically located between the Czech Republic and France



ASSISTANT

Germany is a country that is physically located  
between the Czech Republic and France.



USER

Is Germany physically located between the Czech Republic and France?



ASSISTANT

No, Germany is not physically located between the Czech Republic and France. The Czech Republic is located to the east of Germany, while France is located to the west of Germany. Germany shares borders with both countries, but it is not geographically situated between them.

# Problems are still easy to find



USER

Name me a country that is physically located between the Czech Republic and France



ASSISTANT

Germany is a country that is physically located  
between the Czech Republic and France.



USER

Is Germany physically located between the Czech Republic and France?



ASSISTANT

No, Germany is not physically located between the Czech Republic and France. The Czech Republic is located to the east of Germany, while France is located to the west of Germany. Germany shares borders with both countries, but it is not geographically situated between them.

Models can be wildly inconsistent, hallucinate, prompting is unstable (different results based on small changes).

# Problems are still easy to find



USER

Is Germany physically located between the Czech Republic and France?



ASSISTANT

Yes, Germany is physically located between the Czech Republic and France. The Czech Republic is to the east of Germany, while France is to the west.



USER

Are you sure



ASSISTANT

I apologize for the mistake in my previous response. Germany is not physically located between the Czech Republic and France. The Czech Republic is to the east of Germany, while France is to the west of Germany. Thank you for pointing that out.

Models have been improving fast, sort of...

# Problems are still easy to find



USER

Is Germany physically located between the Czech Republic and France?



ASSISTANT

Yes, Germany is physically located between the Czech Republic and France. The Czech Republic is to the east of Germany, while France is to the west.



USER

## Problem: Models have been improving at rapid speed



ASSISTANT

I apologize for the mistake in my previous response. Germany is not physically located between the Czech Republic and France. The Czech Republic is to the east of Germany, while France is to the west of Germany. Thank you for pointing that out.

Models have been improving fast, sort of...

# Problems are still easy to find



USER

Name me a country that is physically located between the Czech Republic and France



ASSISTANT

Germany is a country that is physically located  
between the Czech Republic and France.



USER

Is Germany physically located between the Czech Republic and France?

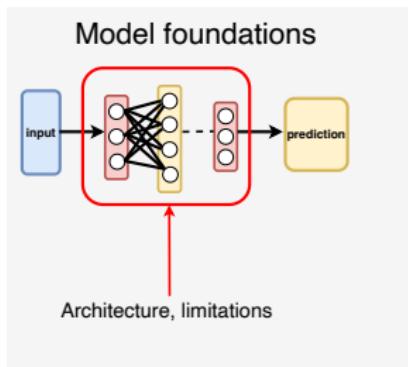


ASSISTANT

No, Germany is not physically located between the Czech Republic and France. The Czech Republic is located to the east of Germany, while France is located to the west of Germany. Germany shares borders with both countries, but it is not geographically situated between them.

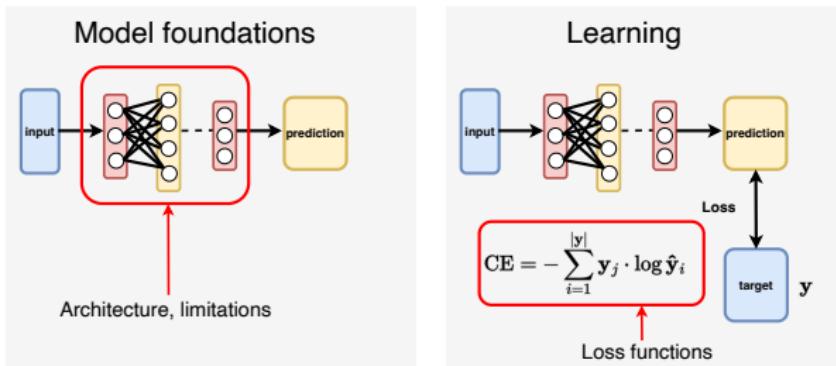
- ▶ Can programming techniques be useful here? constraint programming, probabilistic and logical programming...

# The landscape of NLP research



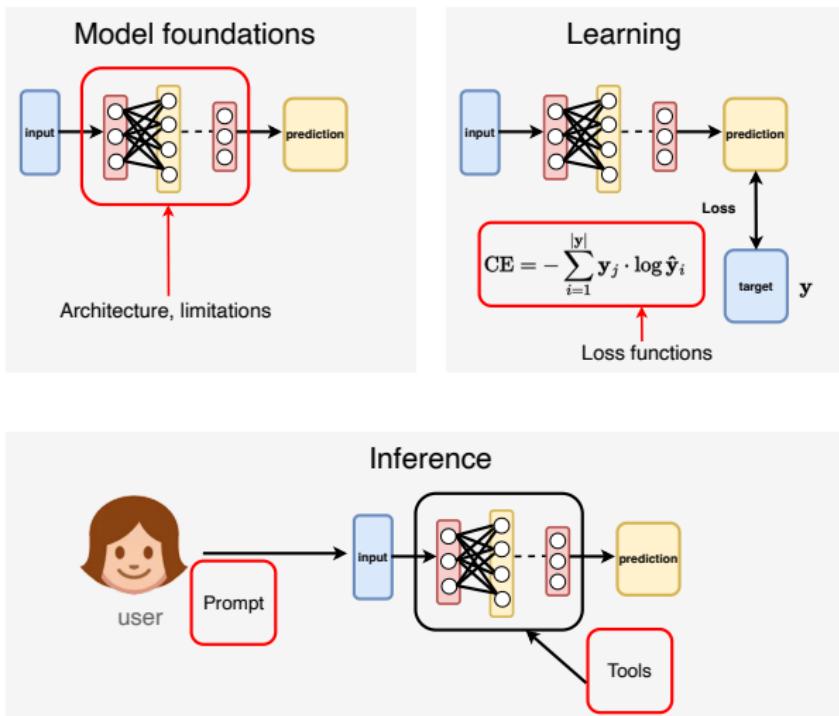
**What model to use?** What kinds of computational problems can models solve? Limitations

# The landscape of NLP research



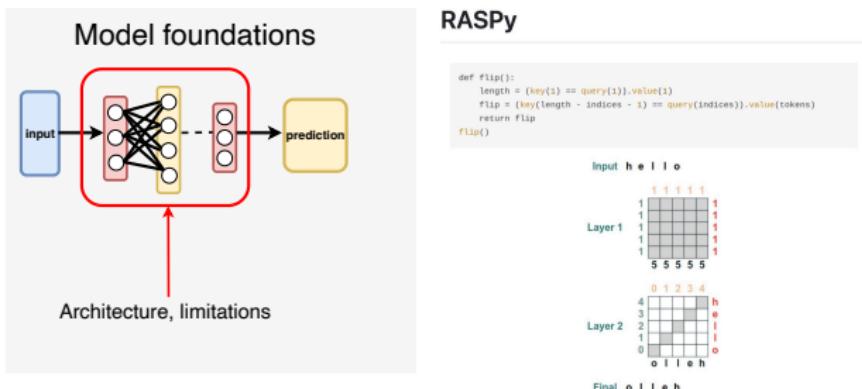
How to train and tune models? Can we design novel loss functions, constraints? (image adapted from <https://www.v7labs.com/blog/pytorch-loss-functions>).

# The landscape of NLP research

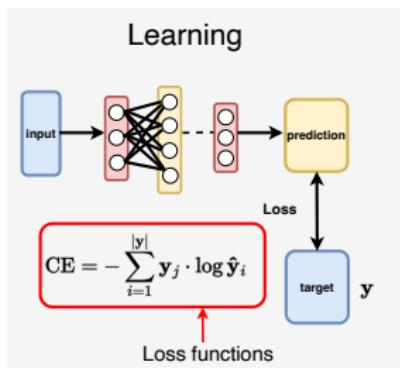


**How to use models?** Decoding with constraints, advanced prompting strategies, coupling with tools.

# Where programming comes in



# Where programming comes in

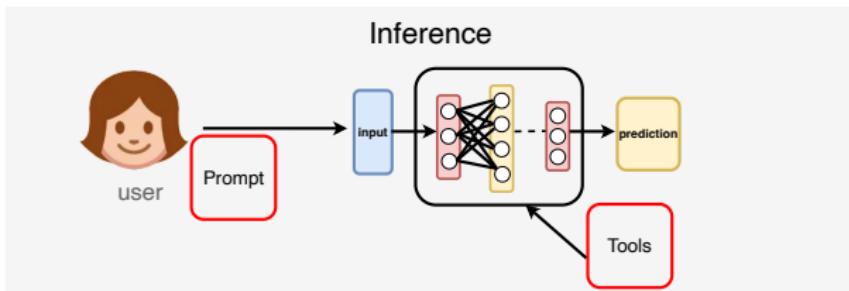


```
1 // File path_planner.scl
2 type actor(x: i32, y: i32), goal(x: i32, y: i32), enemy(x: i32, y: i32)
3
4 const UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3
5 rel safe_cell(x, y) = range(0, 5, x), range(0, 5, y), not enemy(x, y)
6 rel edge(x, y, x, yp, UP) = safe_cell(x, y), safe_cell(x, yp), yp == y + 1
7 // Rules for DOWN, RIGHT, and LEFT edges are omitted...
8
9 rel next_pos(p, q, a) = actor(x, y), edge(x, y, p, q, a)
10 rel path(x, y, x, y) = next_pos(x, y, _)
11 rel path(x1, y1, x3, y3) = path(x1, y1, x2, y2), edge(x2, y2, x3, y3, _)
12 rel next_action(a) = next_pos(p, q, a), path(p, q, r, s), goal(r, s)
```

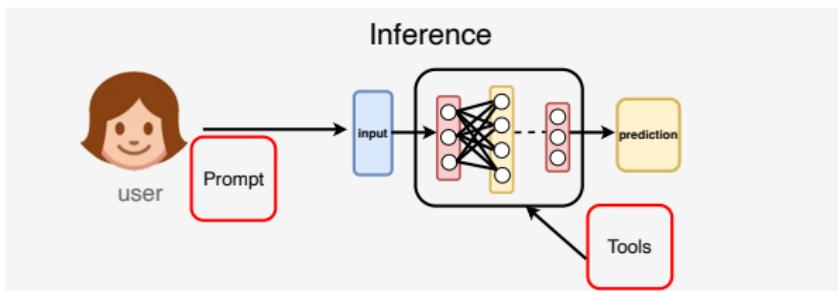
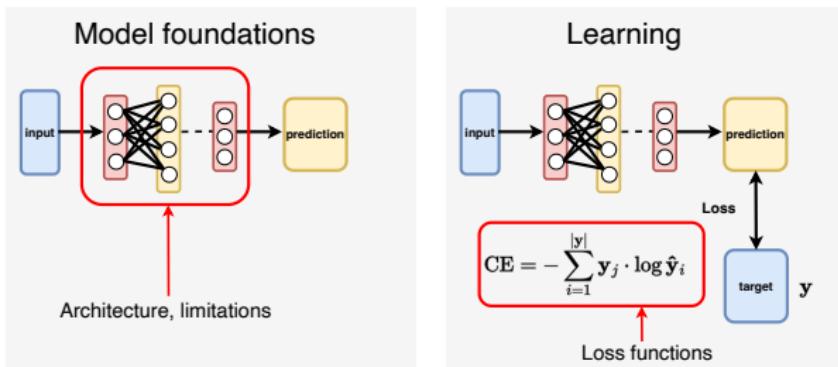
Fig. 3. The logic program of the PacMan-Maze application in Scallop.

# Where programming comes in

```
●●●  
@lmql.query  
def meaning_of_life():  
    """  
        # top-level strings are prompts  
        "Q: What is the answer to life, the \  
        universe and everything?"  
  
        # generation via (constrained) variables  
        "A: [ANSWER]" where \  
            len(ANSWER) < 120 and STOPS_AT(ANSWER, ".")  
  
        # results are directly accessible  
        print("LLM returned", ANSWER)  
  
        # use typed variables for guaranteed  
        # output format  
        "The answer is [NUM: int]"  
  
        # query programs are just functions  
        return NUM  
    ...  
  
    # so from Python, you can just do this  
meaning_of_life() # 42
```



# Thematic overview



**warning:** will only cover a small space, is a broad landscape, focus on underlying techniques and applications.

# Schedule

# Schedule

- ▶ **Lecture 1: Language model basics** and background, symbolic programming of transformers.

# Schedule

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for model fine-tuning, semantic loss, logic background.

# Schedule

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for model fine-tuning, semantic loss, logic background.
- ▶ **Lecture 3:** High-level programming for direct preference alignment, semantics of LLM algorithms.

# Schedule

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for model fine-tuning, semantic loss, logic background.
- ▶ **Lecture 3:** High-level programming for direct preference alignment, semantics of LLM algorithms.
- ▶ **Lecture 4:** Algorithms for test-time inference, logic programming.

# Schedule

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for model fine-tuning, semantic loss, logic background.
- ▶ **Lecture 3:** High-level programming for direct preference alignment, semantics of LLM algorithms.
- ▶ **Lecture 4:** Algorithms for test-time inference, logic programming.
- ▶ **Lecture 5:** Model prompting and imperative (probabilistic) programming, implementing DSLs in Python.

# Schedule

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for **model fine-tuning**, semantic loss, logic background.
- ▶ **Lecture 3:** High-level programming for **direct preference alignment**, semantics of LLM algorithms.
- ▶ **Lecture 4:** Algorithms for **test-time inference**, logic programming.
- ▶ **Lecture 5:** **Model prompting** and imperative (probabilistic) programming, implementing DSLs in Python.

*Not covered:* Programming for **constrained decoding** (*Gij*s).

## Assumed background

- ▶ **Lecture 1:** Language model basics and background, symbolic programming of transformers.
- ▶ **Lecture 2:** Declarative programming for model fine-tuning, semantic loss, logic background.
- ▶ **Lecture 3:** High-level programming for direct preference alignment, semantics of LLM algorithms.
- ▶ **Lecture 4:** Algorithms for test-time inference, logic programming.
- ▶ **Lecture 5:** Model prompting and imperative (probabilistic) programming, implementing DSLs in Python.

*Prerequisites:* Basic knowledge of machine learning, Pytorch, logic.

# Technical foundations of course

Logic background

**Kyle Richardson**

Allen Institute for Artificial Intelligence (AI2)

July 2025

# Code examples and useful libraries

```
1  ### installation via pip
2  ### pip install install z3-solver python-sat torch
3  # datasets transformers sympy PySDD pylon-lib
4  # numpy problog
5
6  ### neural network
7  import torch ## tensor computation, deep learning
8  import transformers ### transformer models
9
10 ### solvers and theorem provers
11 import z3-solver ## z3 solver
12 import pysat ## python interface to sat solvers
13 import problog ### probabilistic logic programming
14
15 ### other useful utilities
16 import sympy # symbolic computation in python
17 import pysdd # knowledge compilation
18 import numpy ## numerical computation in python
19 import datasets ## huggingface datasets
```

# Code examples and useful libraries

```
1  ### installation via pip
2  ### pip install install z3-solver python-sat torch
3  # datasets transformers sympy PySDD pylon-lib
4  # numpy problog
5
6  ### neural network
7  import torch ## tensor computation, deep learning
8  import transformers ## transformer models
9
10 ### solvers and utilities
11 import z3-solver ## z3 solver
12 import pysat ## python interface to sat solvers
13 import problog ## probabilistic logic programming
14
15 ### other useful utilities
16 import sympy # symbolic computation in python
17 import pysdd # knowledge compilation
18 import numpy ## numerical computation in python
19 import datasets ## huggingface datasets
```

**Try out the code examples!**

# Language modeling primer

## What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

## What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany}$  is between Czechia and France

# What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany}$  is between Czechia and France

$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

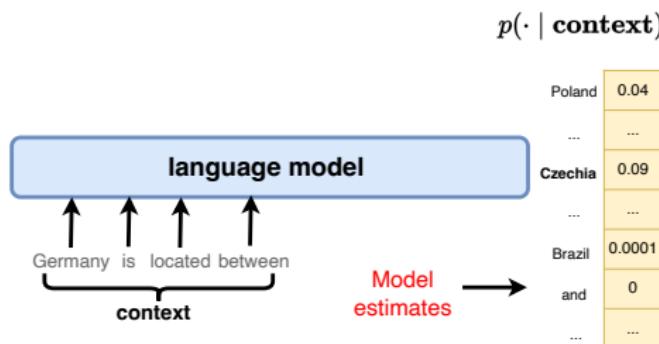
# What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany}$  is between Czechia and France

$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

Language models **generate**



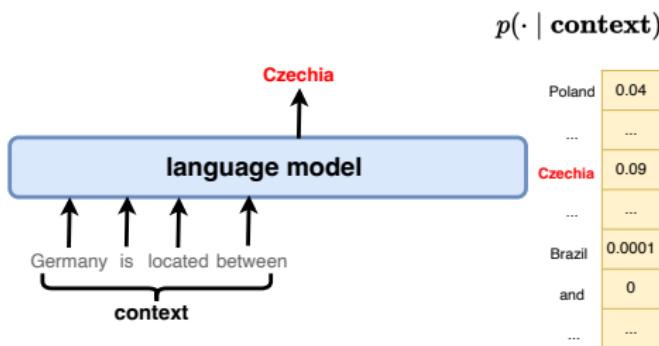
# What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany is between Czechia and France}$

$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

Language models **generate**



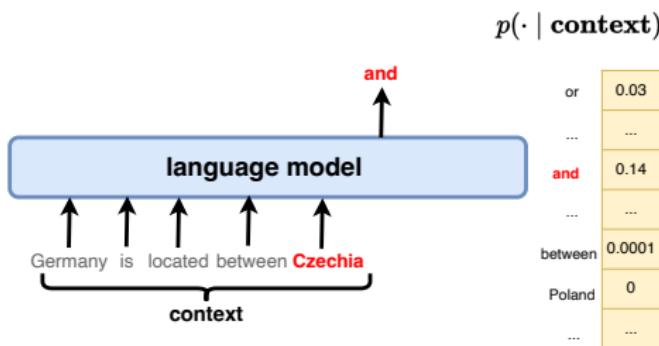
# What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany is between Czechia and France}$

$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

Language models **generate**



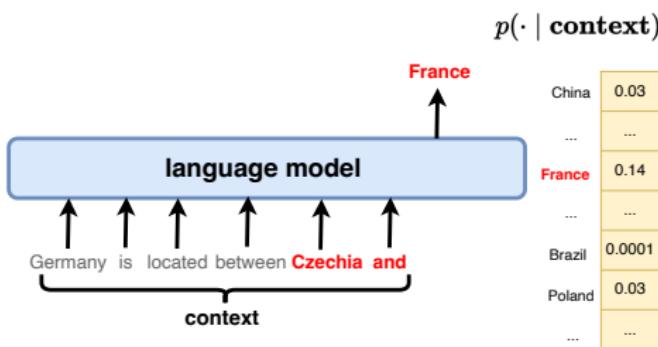
# What are language models?

- ▶ Machine learning models that **assign probabilities to sequences**.

e.g.,  $t = \text{Germany is between Czechia and France}$

$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

Language models **generate**



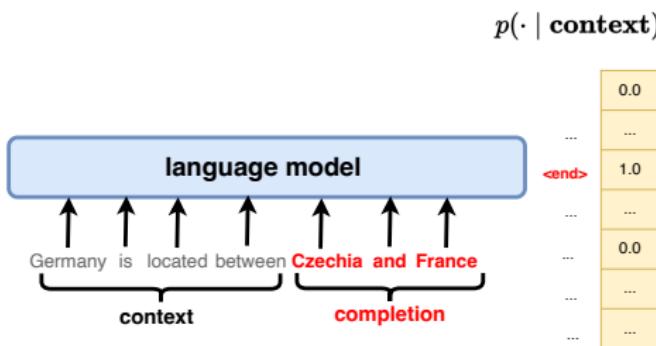
## What are language models?

- ▶ Machine learning models that assign probabilities to sequences.

e.g.,  $t =$ Germany is between Czechia and France

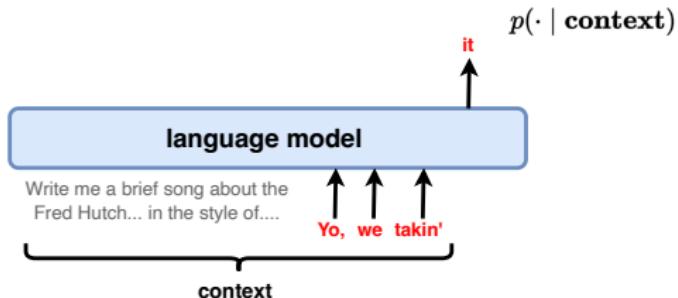
$$\underbrace{p(w_1, w_2, \dots, w_m)}_{\text{joint distribution}} = \prod_{j=1}^m p(\underbrace{w_j}_{\text{next word}} \mid \underbrace{w_1, \dots, w_{j-1}}_{\text{previous words}})$$

## Language models **generate**



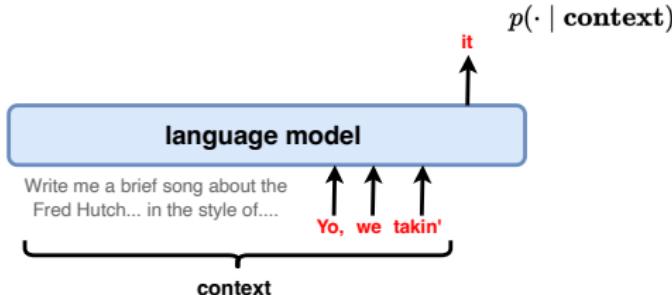
# What are language models?

Sequence modeling is not a new problem.



# What are language models?

Sequence modeling is not a new problem.



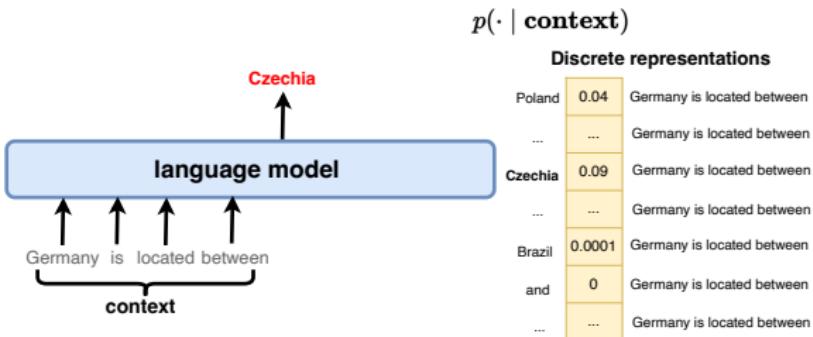
## Text documents

TaskFirst pose the question:Here are four triangles. What do all of these triangles have in common? What makes them different from the figures that are not triangles? What is true for some but not all of these triangles? ##IMAGE0## If students come up with a statement that is true about all of the triangles that they see but not true of all triangles in general, the teacher should ask students if they can imagine a triangle without that attribute. For example, if a student says, "All of the triangles are white on the inside," the teacher can ask, "Would it be possible for a triangle to have a different color on the inside?" When the class comes up with an attribute that is truly shared by all triangles, then the class can complete the sentence frame: All triangles \_\_\_\_\_, but only some triangles \_\_\_\_\_. When the students have written (or composed) their sentences based on the sentence frames, the class can write the definition of a triangle together:A triangle is a closed shape w

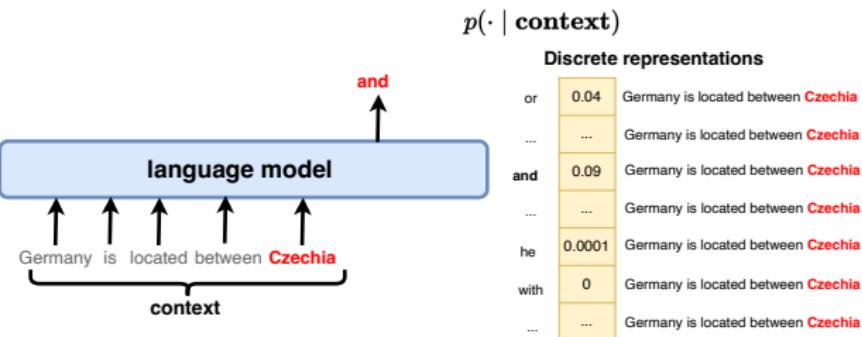
## Biological sequences

```
ATG GAT TTT GGT GTT TGT CTT CTG CCT GTG TGG GTT GAT GGT TTG GAG GGT  
GCG GAG TAG CCG GCA GCG GAT GAG CGG CGC CGG TCC GTC GCA GCG GGA GGA  
GGA CCC ATG  
GAA GCG GAA GGC GCT GCC CGC CGG CGT CCA CGC CGT GAG GCC ATC GAG  
GGA GGC GGT GGC GAC CCA CCA ACA CGC GAT GCC AGT TTG ACC GGC AGC CGC GGG  
CGA CGA CGG  
CTT GCC AAA GGC CGG GGC TGC CGG GCG CGC CGG GCT CTG GTG GAG GAG GGC  
CGA GAG GAA CTT GGC TTC CGA CGC CGC CAC CTG GCG GGC TGC GGA GCG GCC CTG  
CGT CGA CGC  
ACT CGG CGT GGC CGC GAG GCG GAG ATG TGC CGC GCG CGT GAG GAC GGC ATC  
GCC AGC GAA CTG GAG GTG CGC AGA TGG TGC CGG GAT GAG CGG TGC CGC GGC CTG  
GCC CCT
```

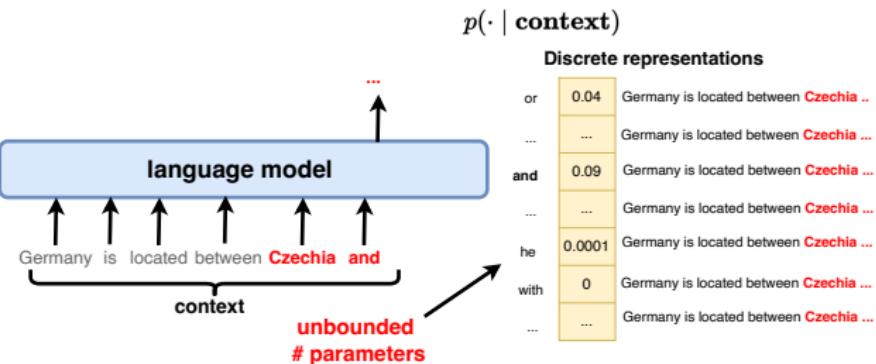
# The problem with traditional solutions



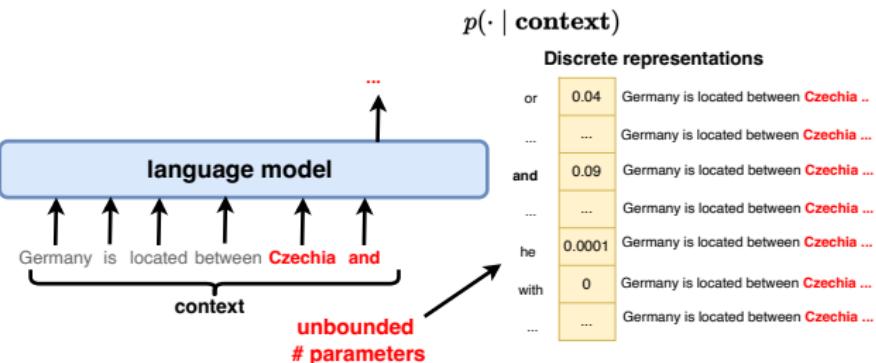
# The problem with traditional solutions



# The problem with traditional solutions

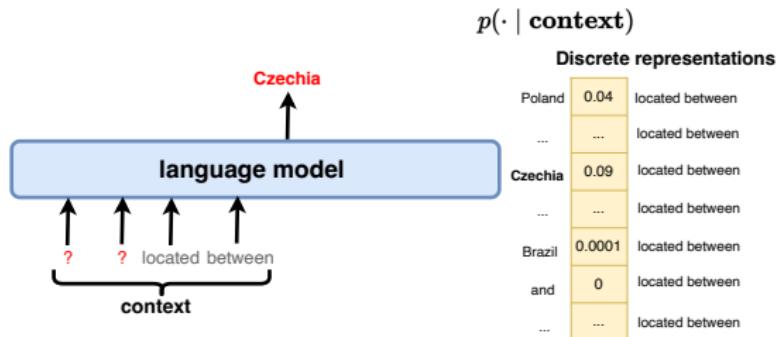


# The problem with traditional solutions



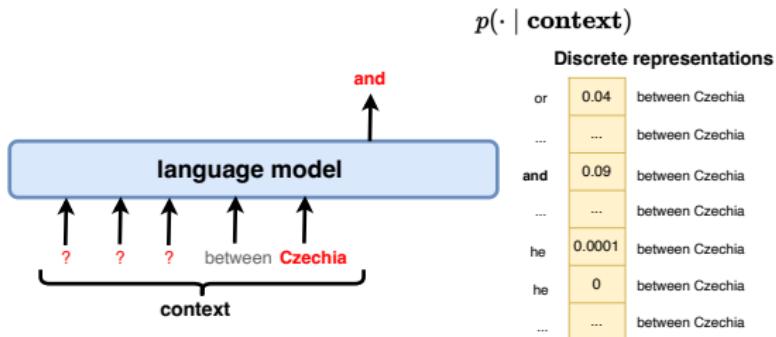
Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

# The problem with traditional solutions



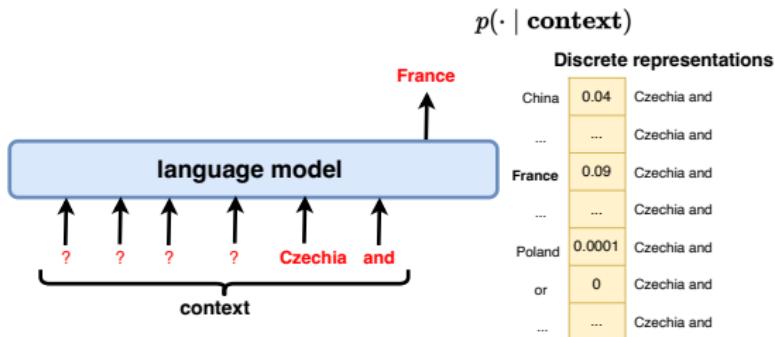
Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

# The problem with traditional solutions



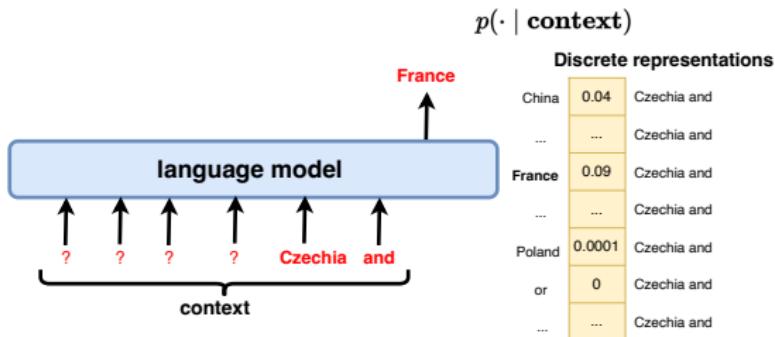
Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

# The problem with traditional solutions



Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

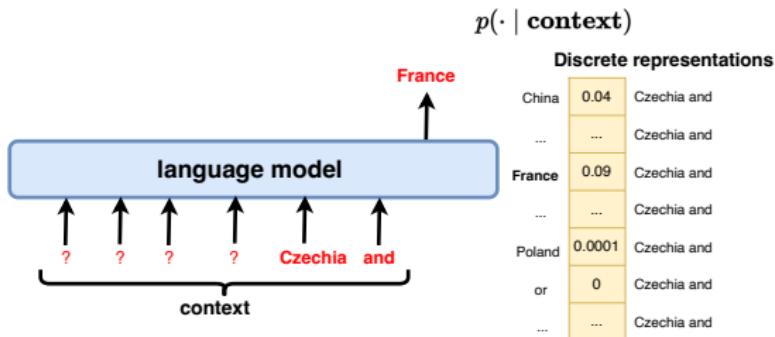
# The problem with traditional solutions



Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

Even limited to 2 previous words and a vocabulary of 100,000 words, results in  $10^{15}$  (**quadrillion**) probability parameters.

# The problem with traditional solutions

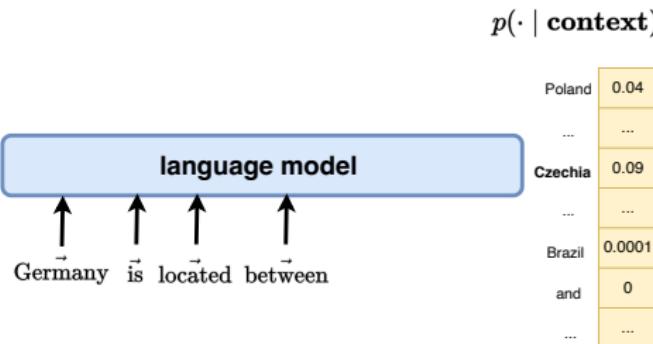


Make models tractable through **independence** or **Markov** assumptions;  
limit the allowable context.

**Do not** faithfully model these joint probability distributions!

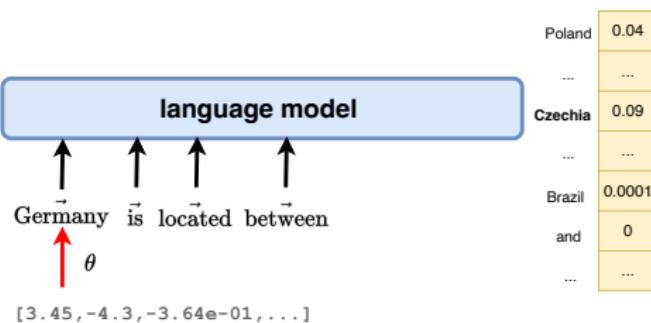
# Moving from the discrete to the continuous

- Modern deep learning systems operate in the continuous space.



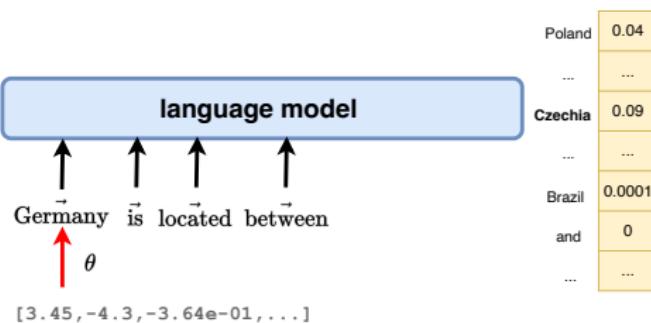
# Moving from the discrete to the continuous

- Modern deep learning systems operate in the continuous space.



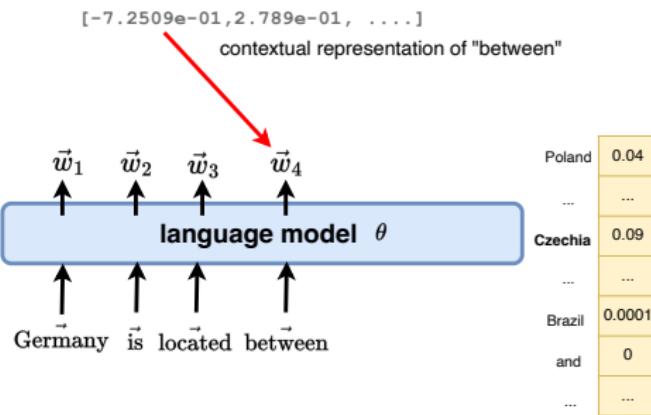
# Moving from the discrete to the continuous

- Modern deep learning systems operate in the continuous space.



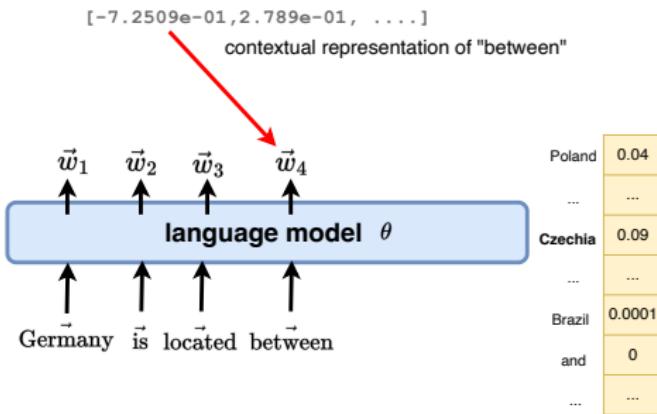
# Moving from the discrete to the continuous

- Modern deep learning systems operate in the continuous space.



# Moving from the discrete to the continuous

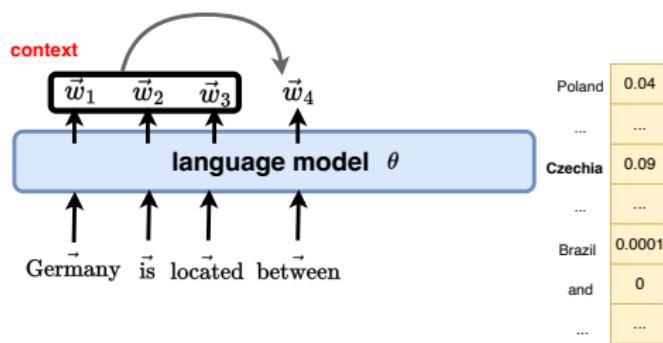
- Modern deep learning systems operate in the continuous space.



How information flows / contextual representations are built, relates to the **model architecture**. Popular: **transformer architecture**.

# Moving from the discrete to the continuous

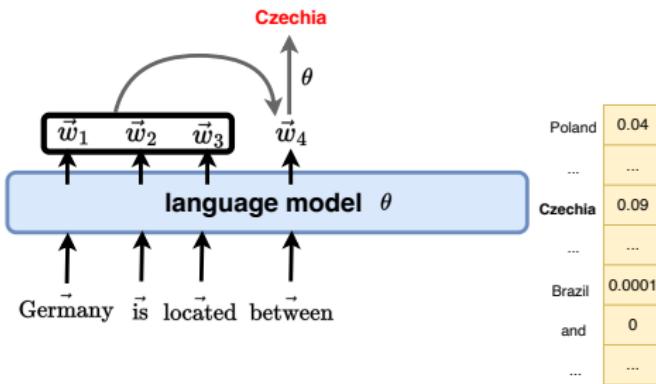
- Modern deep learning systems operate in the continuous space.



How information flows / contextual representations are built, relates to the **model architecture**. Popular: **transformer architecture**.

# Moving from the discrete to the continuous

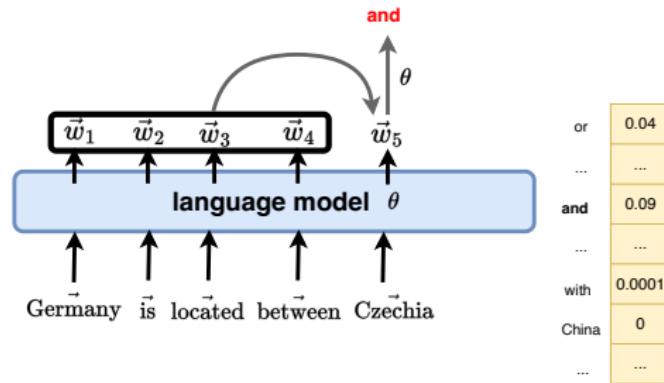
- Modern deep learning systems operate in the continuous space.



**Do not** make independence assumption, can faithfully model these joint probability distributions.

# Moving from the discrete to the continuous

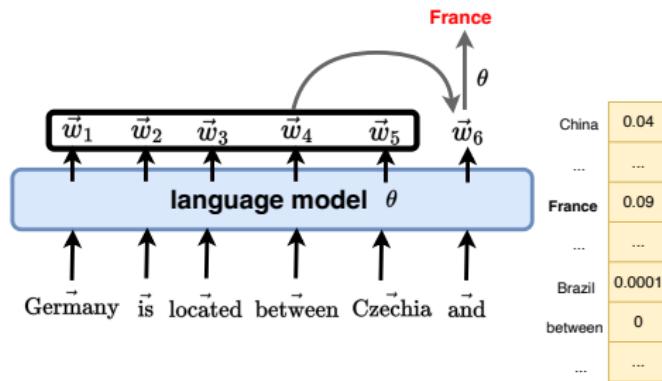
- Modern deep learning systems operate in the continuous space.



**Do not** make independence assumption, can faithfully model these joint probability distributions.

# Moving from the discrete to the continuous

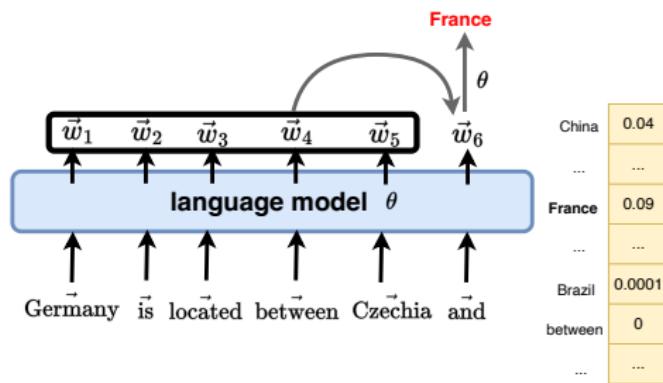
- Modern deep learning systems operate in the continuous space.



**Do not** make independence assumption, can faithfully model these joint probability distributions.

# Moving from the discrete to the continuous

- Modern deep learning systems operate in the continuous space.



**Fundamental problem:** learning these underlying representations.

It took time to get this right...

It took time to get this right...

<b>Input:</b>	I come from <u>Tunisia</u> .
<b>Reference:</b>	<u>チュニジア</u> の 出身です。 Chunisia no shusshindesu.
<b>System:</b>	(I'm from <u>Tunisia</u> .) ノルウェー の 出身です。 Noruue- no shusshindesu. (I'm from <u>Norway</u> .)
	Target translation Mistake

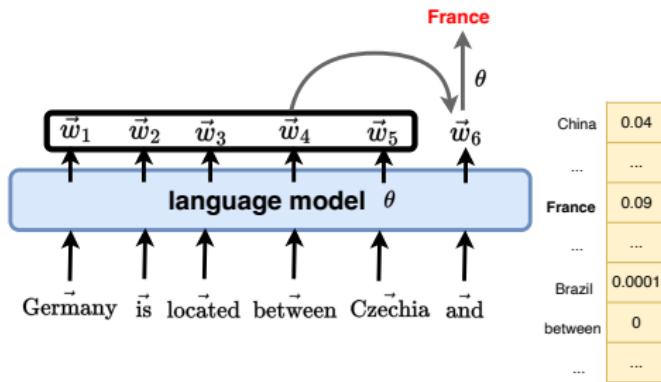
Figure 1: An example of a mistake made by NMT  
on low-frequency content words.

It took time to get this right...

Table 2: Errors from the RNN.

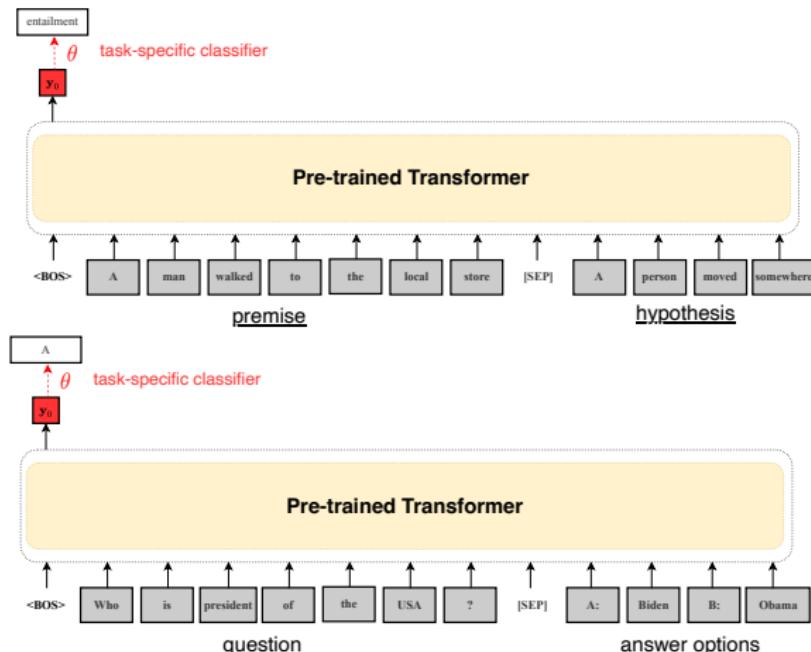
Input	Correct	Prediction
2 mA	two milliamperes	two million liters
11/10/2008	the tenth of november	the tenth of october
1/2 cc	two thousand eight	two thousand eight
18:00:00Z	half a c c	one minute c c
	eighteen hours zero minutes and zero seconds z	eighteen hundred cubic minutes

# (Pre-)Training in a nutshell



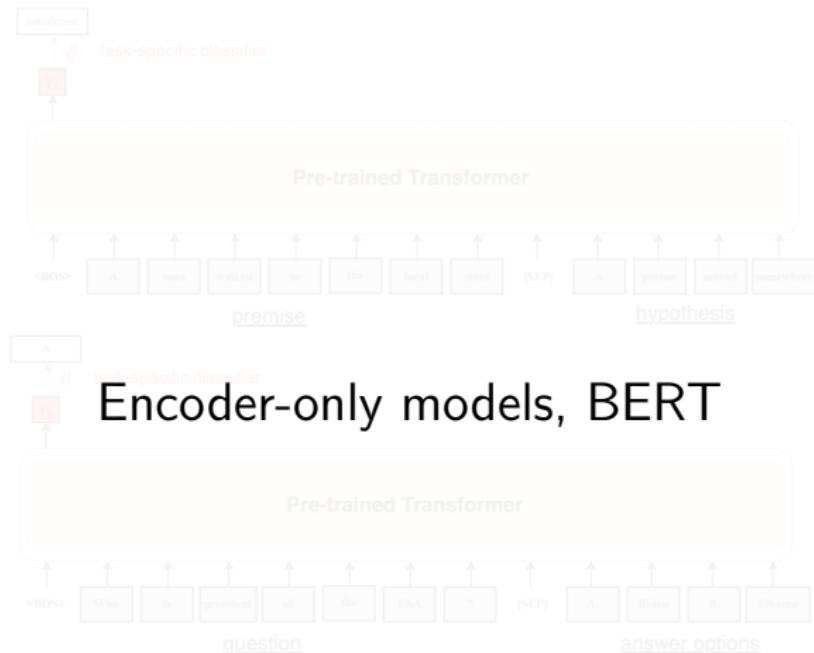
- ▶ Estimate parameters  $\theta$  from signal you get from *auto-completing* example data. Straightforward to get lots of **data for free**.

# Model Fine-tuning



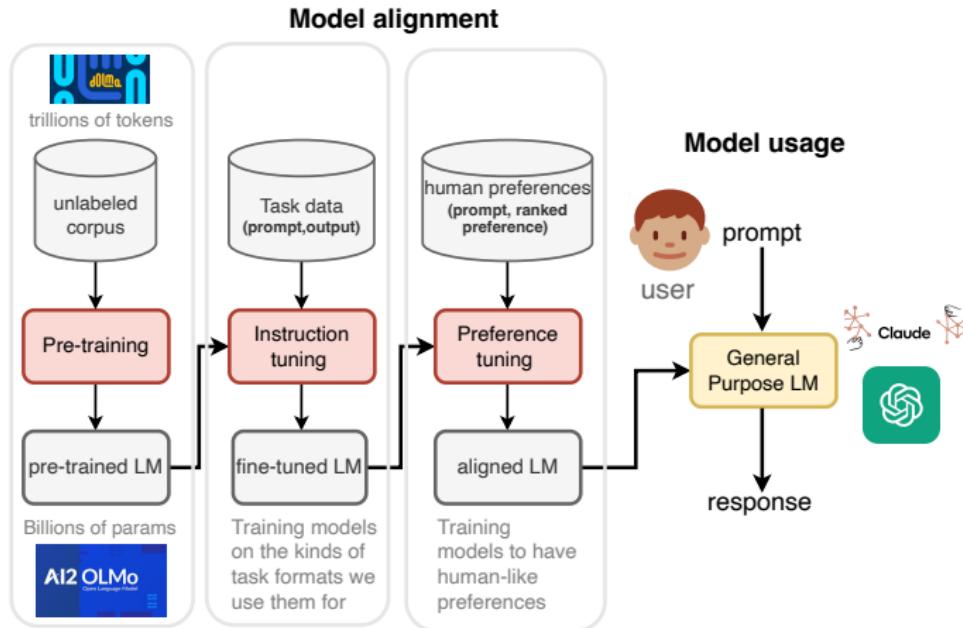
**Fine-tuning:** customizing models to target tasks using additional parameters; **idea:** bootstrap off of pre-training knowledge.

# Model Fine-tuning



**Fine-tuning:** customizing models to target tasks using additional parameters; **idea:** bootstrap off of pre-training knowledge.

# General purpose large language models



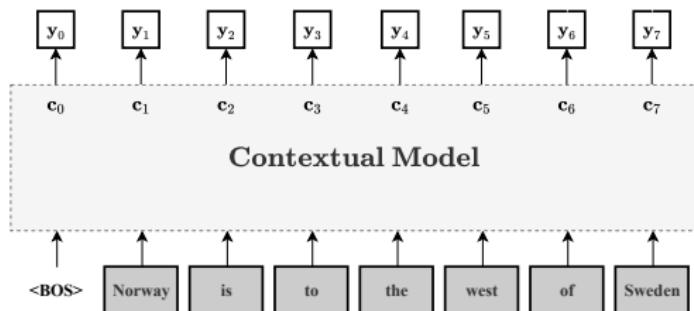
# Contextual models and transformers

## Contextual Models

- ▶ **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.

# Contextual Models

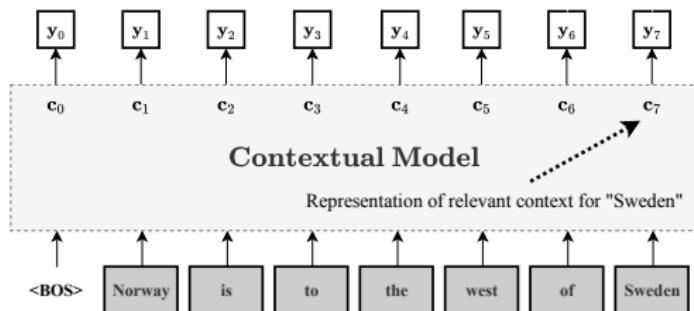
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Operationally:** neural network models, often large and opaque.

# Contextual Models

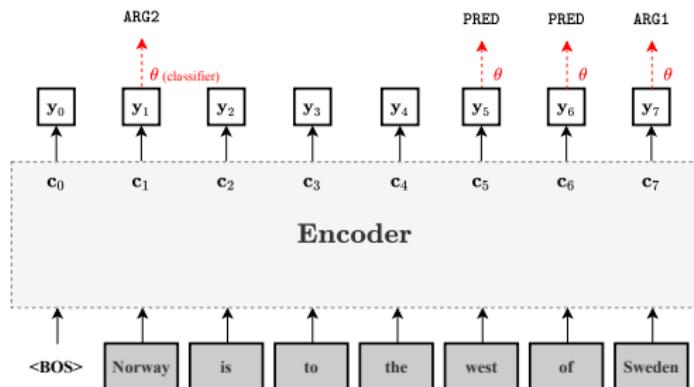
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Semantics:**  $\mathbf{y}_7$  captures the meaning of *Sweden* grounded in this particular sentential context, output of a compositional procedure.

# Contextual Models

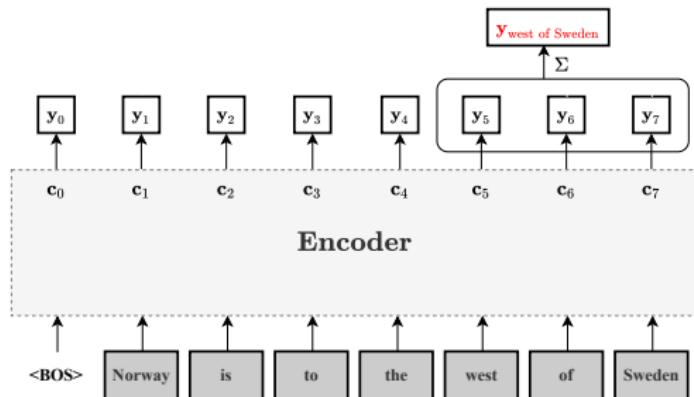
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Why?** Build representations that allow you make predictions, **success:** learned representations that effectively solve problems.

# Contextual Models

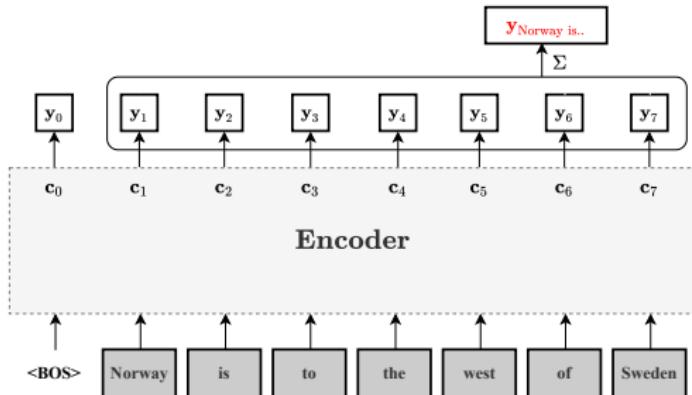
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Why?** Build representations that allow you make predictions, **success:** learned representations that effectively solve problems.

# Contextual Models

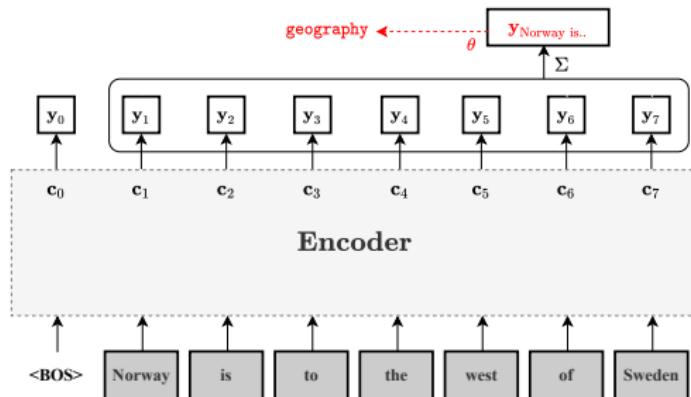
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Why?** Build representations that allow you make predictions, **success:** learned representations that effectively solve problems.

# Contextual Models

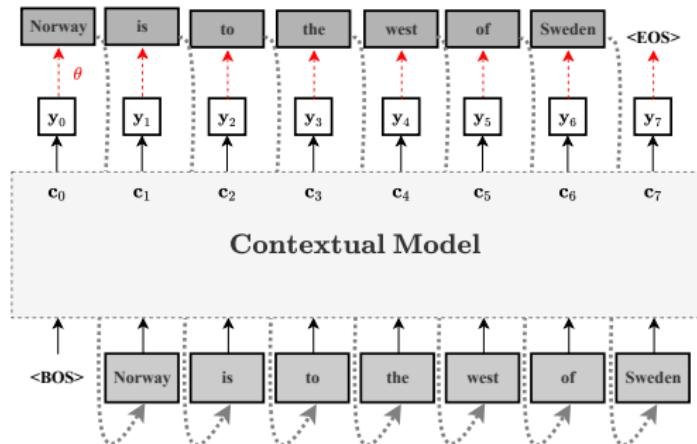
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Why?** Build representations that allow you make predictions, **success:** learned representations that effectively solve problems.

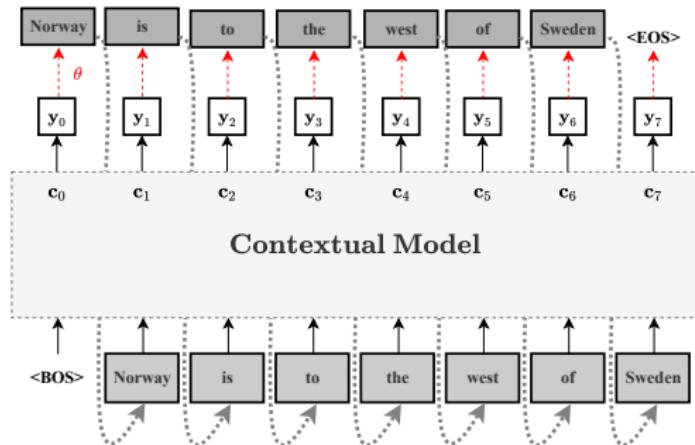
# Contextual Models

- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



# Contextual Models

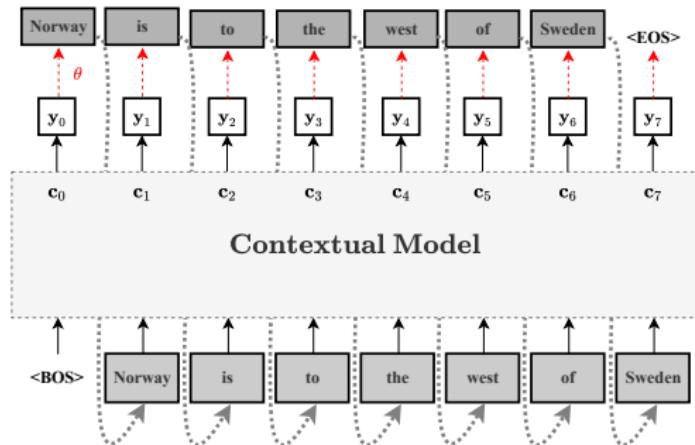
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



as LMs:  $p(w_j | w_1, \dots, w_{j-1}) = p(w_j | c_{j-1})$ , Important: Can condition on full contexts, faithfully model complex joint probability distributions

# Contextual Models

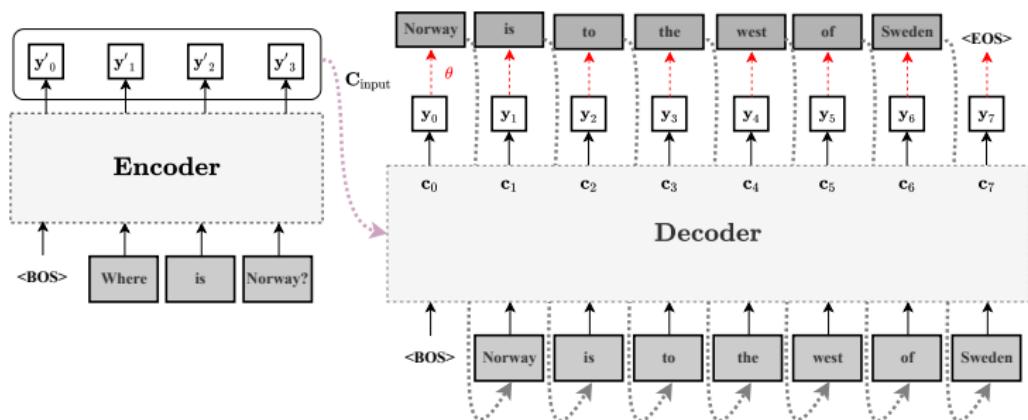
- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.



**Model Architectures:** how information is processed, internal representations are constructed. **Common:** RNNs, Transformers.

# Contextual Models

- **Role:** assign continuous vectors representations  $\mathbf{y}_j \in \mathbb{R}^d$  to elements in a sequence that capture their meaning of those elements in context.

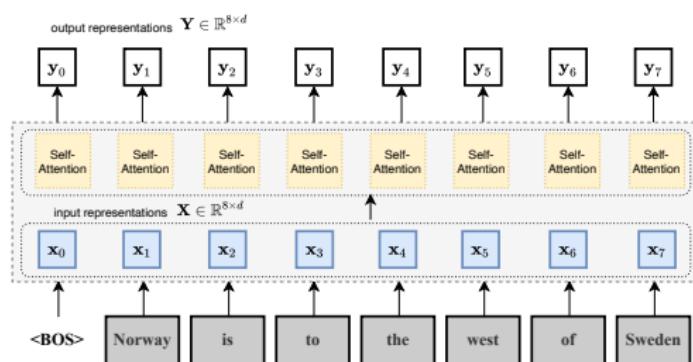


**Text2Text models:** estimate  $p(y^{\text{output}} | x^{\text{input}})$ , natural way to express many language understanding problems.

# Transformers and attention

# Self-Attention: Simple Encoder Example

**Attention:** Mechanism for building contextual representations (see [Vaswani et al. \(2017\)](#)).

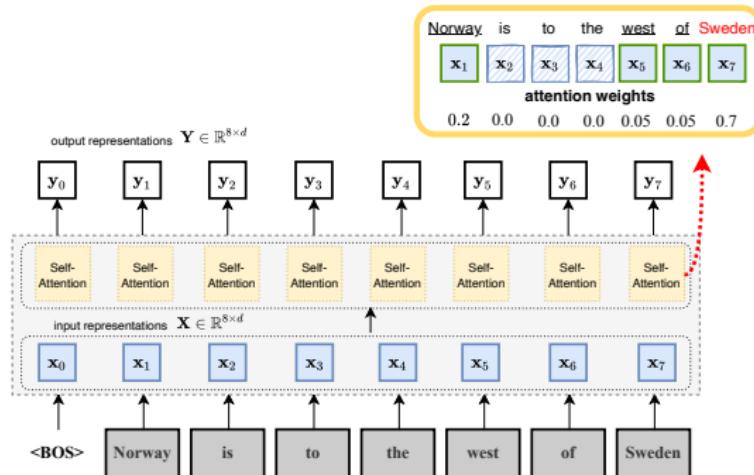


# Self-Attention: Embeddings Representations

```
1 import torch ## to install: pip install pytorch
2
3 ### word embedding parameters and matrix E
4 E = torch.nn.Embedding(
5     embedding_dim=768, ##<--- dimensionality
6     num_embeddings=3000, ##<--- # words
7 )
8
9 ##e.g., Representation of our Input
10 X = E(torch.tensor([
11     0, # <BOS>
12     1, # Norway
13     2, # is
14     3, # to
15     4, # the
16     5, # west
17     6, # of
18     7, # Sweden
19 ])) ### => matrix 7 * 768
20
21 x1 = X[1] ##-> initial representation of Norway
22 print(x1)
23 ##### [-1.1344e+00, -3.0359e-01, 7.3585e-02, ....]
```

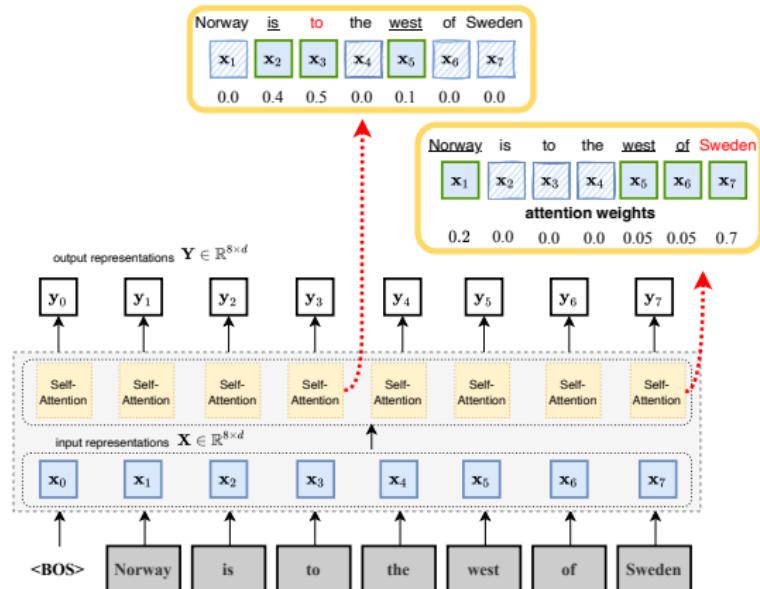
# Self-Attention: Simple Encoder Example

**Attention:** Mechanism for building contextual representations (see [Vaswani et al. \(2017\)](#)).



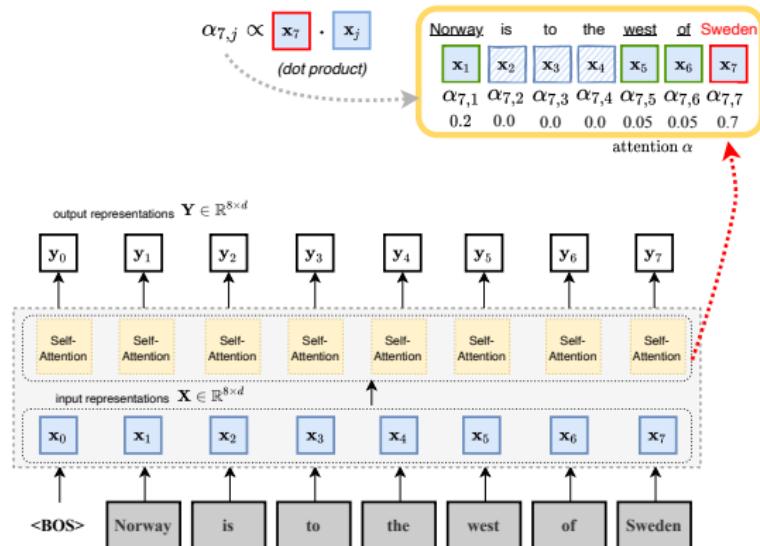
# Self-Attention: Simple Encoder Example

**Attention:** Mechanism for building contextual representations (see [Vaswani et al. \(2017\)](#)).



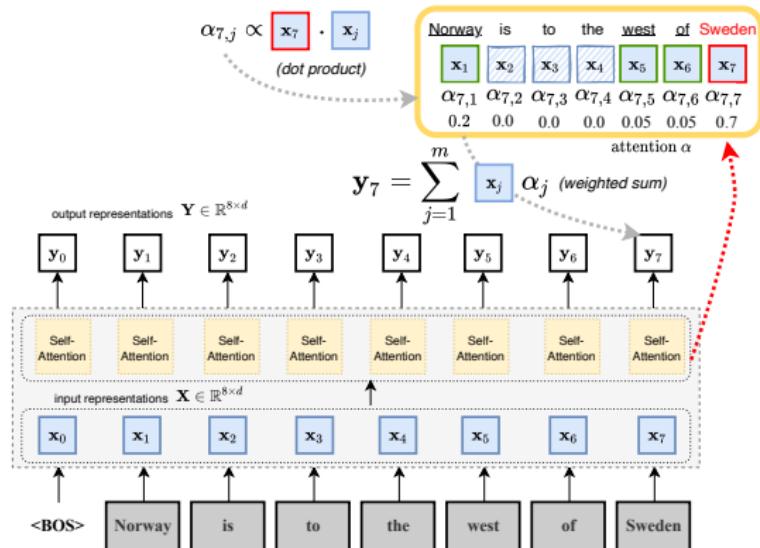
# Self-Attention: Simple Encoder Example

**Attention:** Mechanism for building contextual representations (see [Vaswani et al. \(2017\)](#)).



# Self-Attention: Simple Encoder Example

**Attention:** Mechanism for building contextual representations (see [Vaswani et al. \(2017\)](#)).



# Self-Attention: Computing Final Representations

The full computation

$$\alpha'_{i,j} = \mathbf{x}_i \cdot \mathbf{x}_j \quad \text{dot product}$$

$$\alpha_{i,j} = \frac{e^{\alpha'_{i,j}}}{\sum_j e^{\alpha'_{i,j}}} \quad \text{softmax}$$

$$\mathbf{y}_i = \sum_j \alpha_{i,j} \mathbf{x}_j \quad \text{weighted sum}$$

# Self-Attention: Computing Final Representations

The full computation

$$\alpha'_{i,j} = \mathbf{x}_i \cdot \mathbf{x}_j \quad \text{dot product}$$

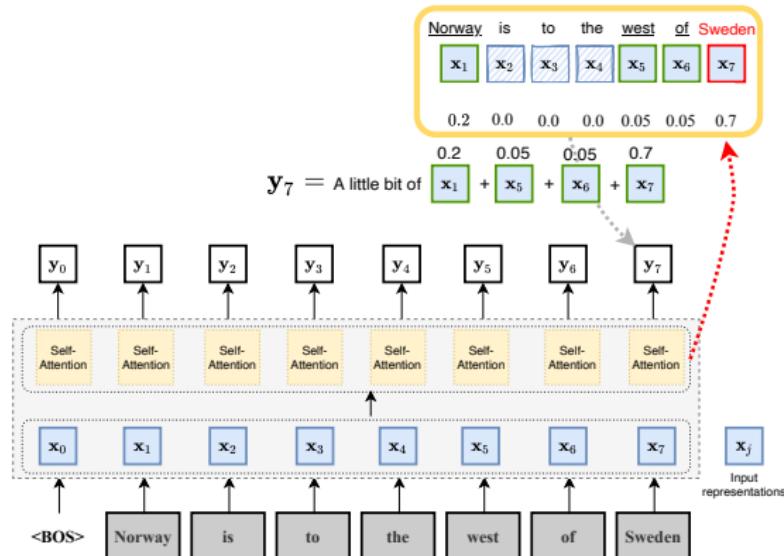
$$\alpha_{i,j} = \frac{e^{\alpha'_{i,j}}}{\sum_j e^{\alpha'_{i,j}}} \quad \text{softmax}$$

$$\mathbf{y}_i = \sum_j \alpha_{i,j} \mathbf{x}_j \quad \text{weighted sum}$$

```
1 ## Input representations (again),
2 X = torch.tensor([0,1,2,3,4,5,6,7])
3
4 ### raw weights (dot product / matrix multiplication)
5 raw_Alpha = torch.matmul(X,X.transpose(0,1))
6 ### normalized via softmax, probability distribution
7 alpha = raw_Alpha.softmax(dim=-1)
8 ### Final self attention representations
9 Y = torch.matmul(alpha,X)
10 ### self attention representation of 'Norway'
11 y1 = Y[1]
```

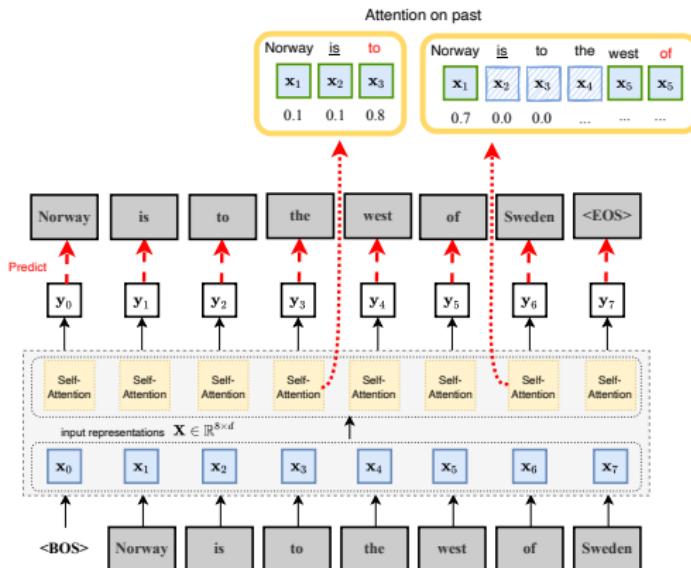
# Self-Attention: Intuition

**Attention:** A kind of brute-force looking around and aggregation of contextual information.



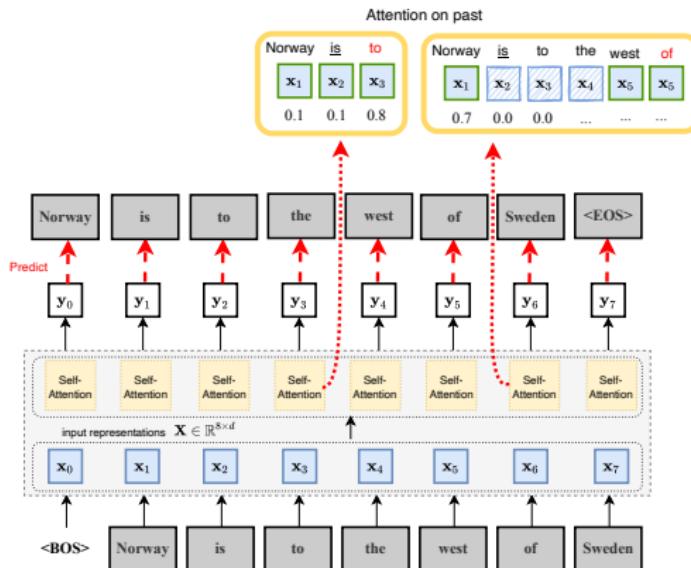
# Attention in Other Contexts

**Decoders:** Attention limited to past context, allows for generation (step-by-step prediction of next word).



# Attention in Other Contexts

**Decoders:** Attention limited to past context, allows for generation (step-by-step prediction of next word).



**Note:** No independence assumptions, full context.

# Causal Attention

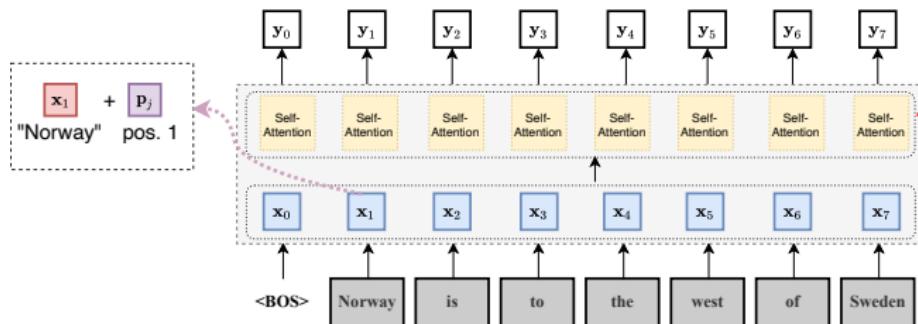
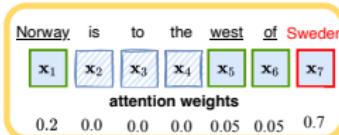
- ▶ Implementing causal attention, involves adding a **mask**.

```
1 import torch
2 ## Input representations (again),
3 X = E(torch.tensor([0,1,2,3,4,5,6,7]))
4
5 ### raw weights (dot product / matrix multiplication)
6 raw_Alpha = torch.matmul(X,X.transpose(0,1))
7
8 ### causal mask
9 mask = torch.triu_indices(8, 8, offset=1)
10 raw_Alpha[:, mask[0], mask[1]] = float('-inf')
11
12 ### normalized via softmax, probability distribution
13 alpha = raw_Alpha.softmax(dim=-1)
14 ### Final self attention representations
15 Y = torch.matmul(alpha,X)
16 ### self attention representation of 'Norway'
17 y1 = Y[1]
```

# An important detail: positional information

- Word embeddings so far do not encode position information.

$$\begin{aligned} \mathbf{x}_j & \quad \mathbf{E} \in \mathbb{R}^{|words| \times d} \text{ (word embeddings)} \\ \mathbf{p}_j & \quad \mathbf{P} \in \mathbb{R}^{p \times d} \quad \text{(position embeddings)} \\ \mathbf{x}_j & = \mathbf{x}_{id} + \mathbf{p}_j \end{aligned}$$



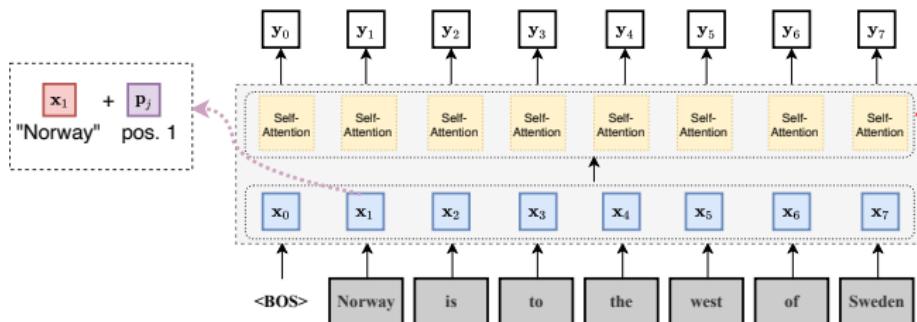
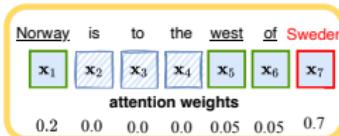
# An important detail: positional information

- Word embeddings so far do not encode position information.

$$\mathbf{x}_j \in \mathbb{R}^{|words| \times d} \text{ (word embeddings)}$$

$$\mathbf{p}_j \in \mathbb{R}^{p \times d} \text{ (position embeddings)}$$

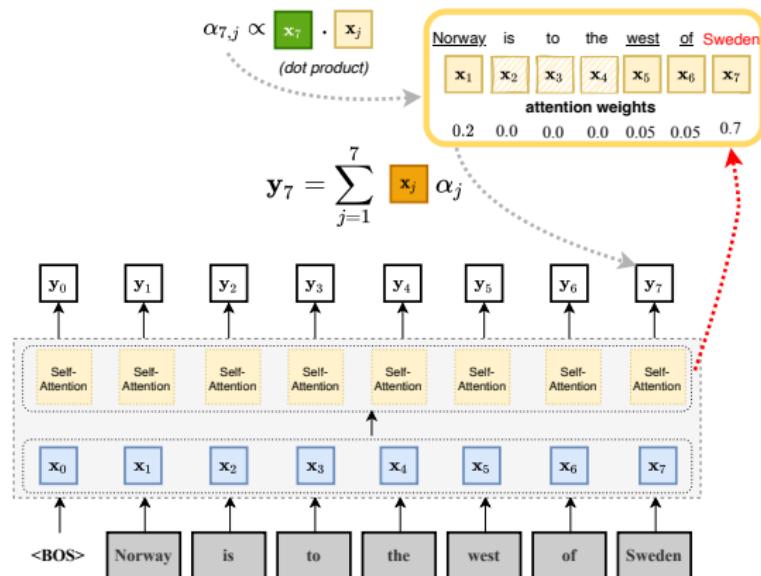
$$\mathbf{x}_j = \mathbf{x}_{id} + \mathbf{p}_j$$



**Note:** we can add any additional information we want, many variations.

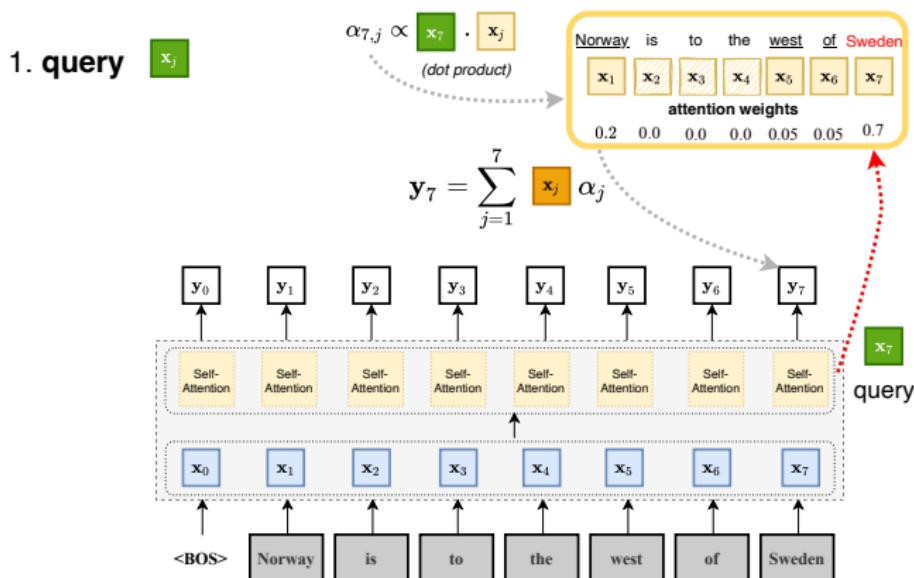
# More Parameters: Keys, Queries and Values

- ▶ Three components in our computation.



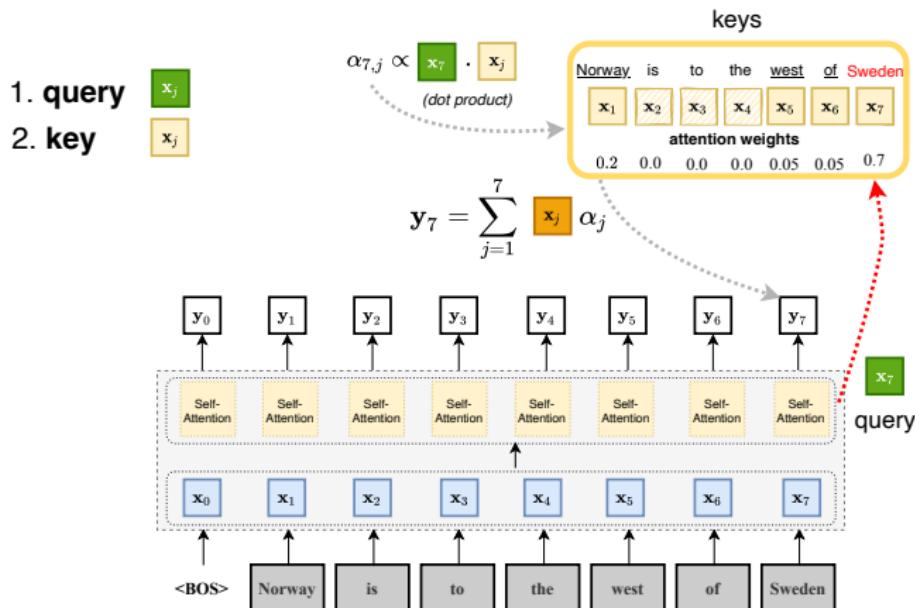
# More Parameters: Keys, Queries and Values

- ▶ Three components in our computation.



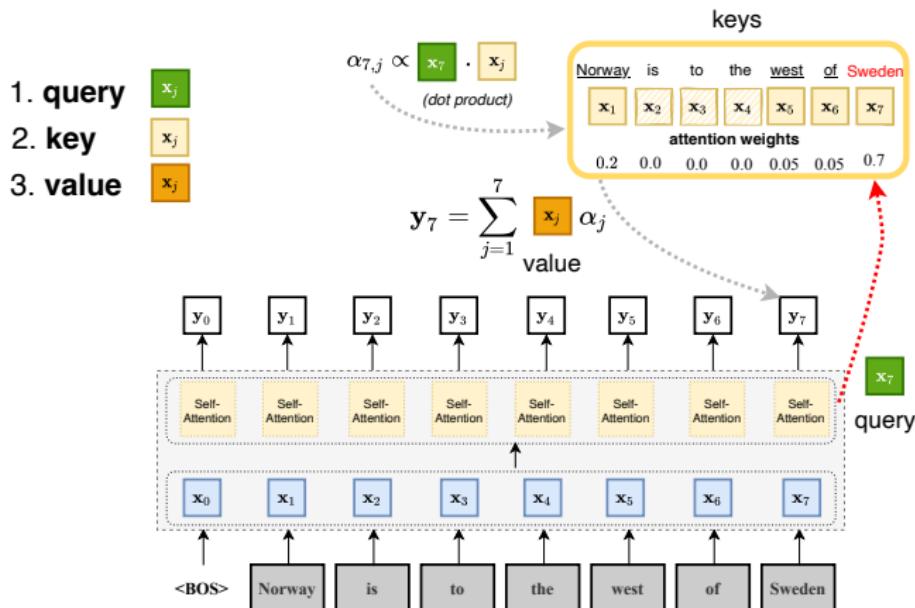
# More Parameters: Keys, Queries and Values

- ▶ Three components in our computation.



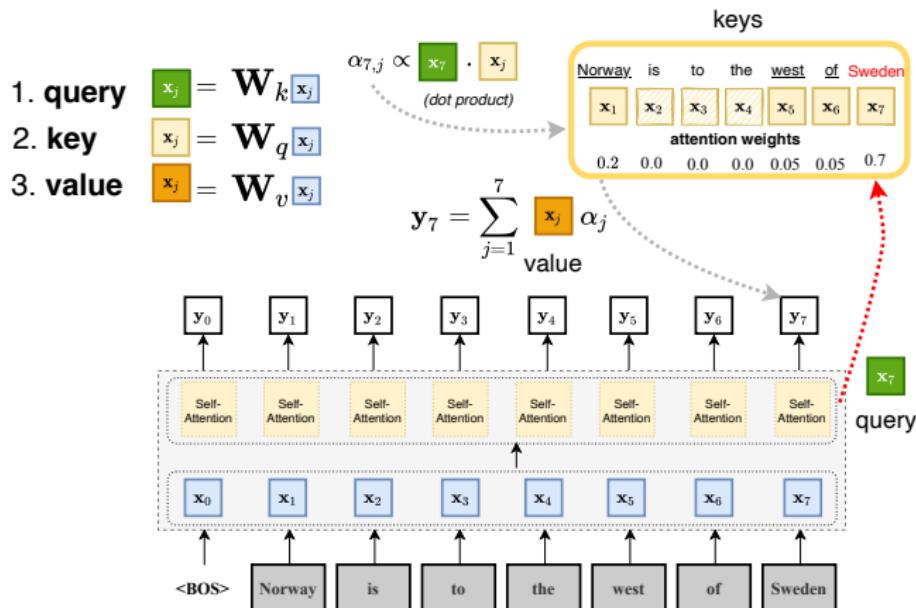
# More Parameters: Keys, Queries and Values

- ▶ Three components in our computation.



# More Parameters: Keys, Queries and Values

- ▶ Three components in our computation.



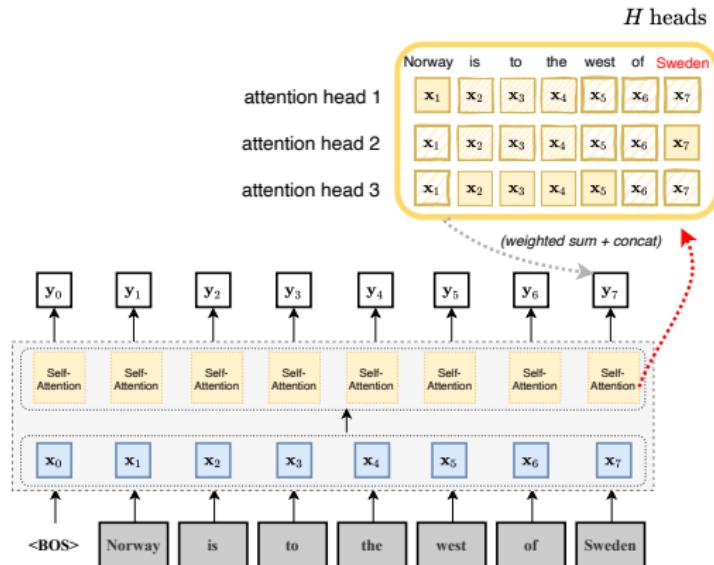
# More Parameters: Keys, Queries and Values

Another few additional lines of PyTorch

```
1 ## Input representations (again),
2 X = E(torch.tensor([0,1,2,3,4,5,6,7]))
3 D = 768
4
5 ### key, value, query parameters (linear layer)
6 W_k    = torch.nn.Linear(D, D, bias=False)
7 W_q    = torch.nn.Linear(D, D, bias=False)
8 W_v    = torch.nn.Linear(D, D, bias=False)
9
10 key_rep   = W_k(X) #<-- rep. of X as keys
11 query_rep = W_q(X) #<-- rep. of X as queries
12 value_rep = W_v(X) #<-- rep of X as values
13
14 ### sample computation with parameters applied over 'X'
15 alpha = torch.matmul(
16     query_rep,
17     key_rep.transpose(0,1)
18 ).softmax(dim=-1)
19
20 ### same as before
21 Y = torch.matmul(alpha,value_rep)
```

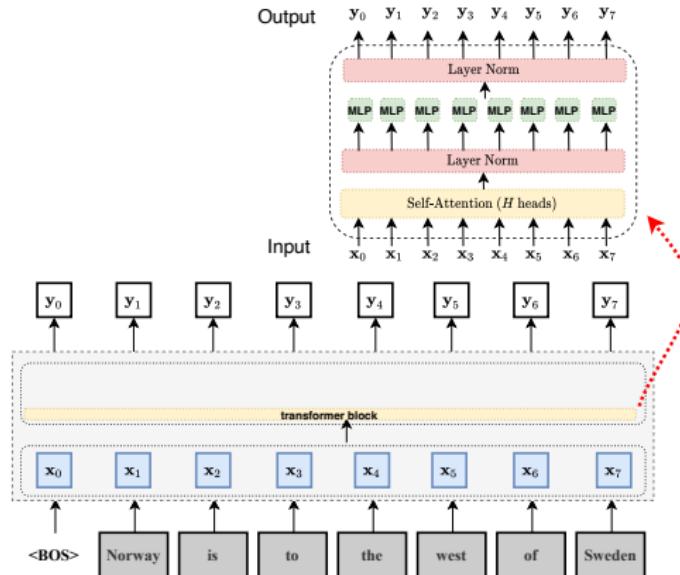
# Multi-headed Attention

- ▶ Allows the model to simultaneously focus on multiple parts of input.



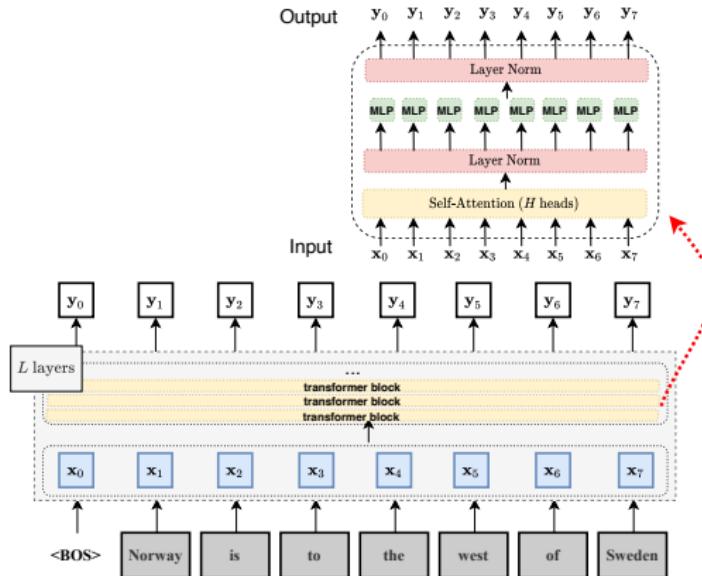
# Transformer Blocks and Multiple Layers

- ▶ Current models are a bit more complex and multi-layered.



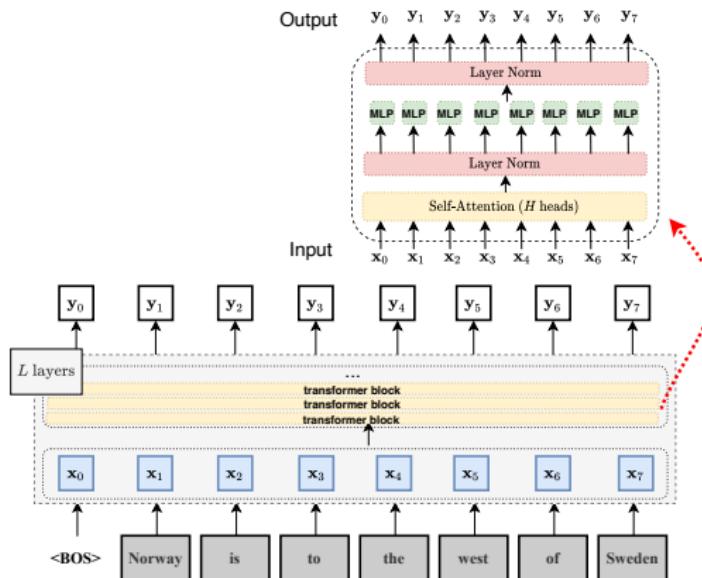
# Transformer Blocks and Multiple Layers

- ▶ Current models are a bit more complex and multi-layered.



# Transformer Blocks and Multiple Layers

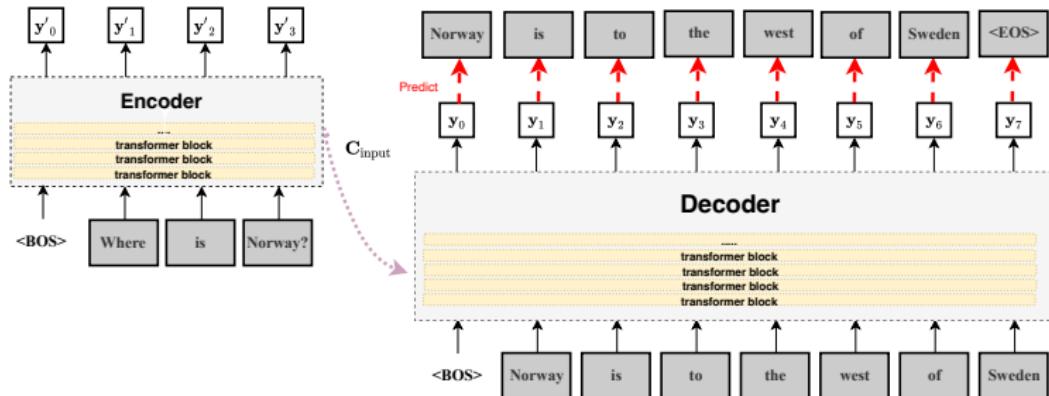
- ▶ Current models are a bit more complex and multi-layered.



**BERT** (Devlin et al., 2019):  $L = 24$  layers each with  $H = 16$  heads,  $340M$  parameters, embedding dimension=1024

# Transformer Blocks and Multiple Layers

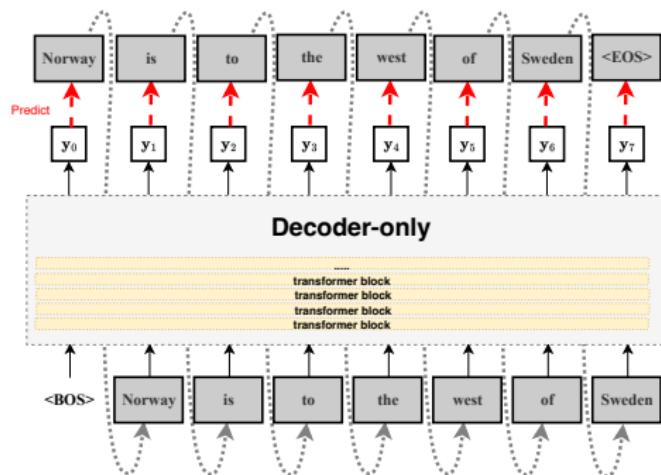
- ▶ Current models are a bit more complex and multi-layered.



T5 model ([Raffel et al., 2020](#)), text2text architecture: **encoder** and **decoder**, 24 layers, 128 heads, cross-attention (11B parameters).

# Transformer Blocks and Multiple Layers

- ▶ Current models are a bit more complex and multi-layered.



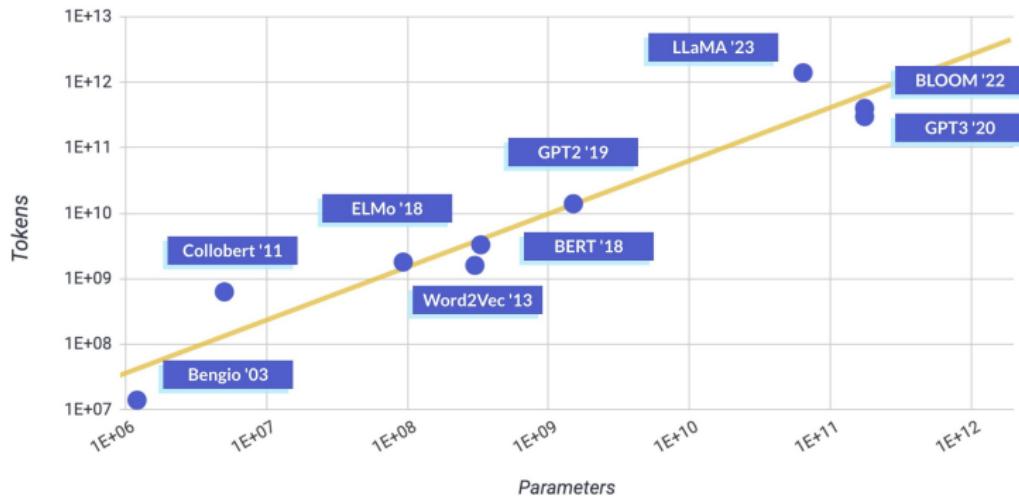
**GPT3** (Brown et al., 2020):  $L = 96$  layers each with  $H = 96$  heads ( $175B$  parameters), embedding dimension=12288

# A typical transformer block

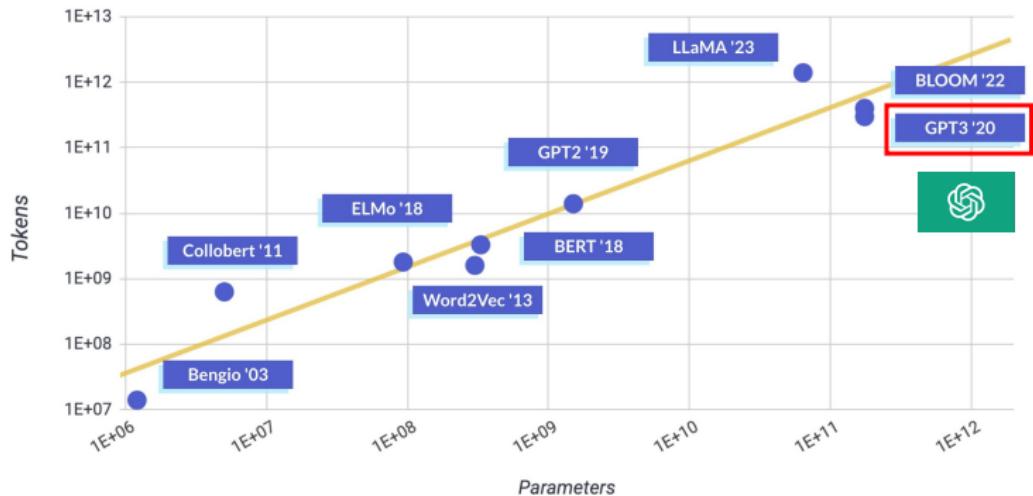
```
1 ## nanoGPT: https://github.com/karpathy/nanoGPT
2 class Block(nn.Module):
3
4     def __init__(self, config):
5         super().__init__()
6         self.ln_1 = LayerNorm(config.n_embd,
7                               bias=config.bias
8         )
9         self.attn = CausalSelfAttention(config)
10        self.ln_2 = LayerNorm(config.n_embd,
11                              bias=config.bias
12        )
13        self.mlp = MLP(config)
14
15    def forward(self, x):
16        x = x + self.attn(self.ln_1(x))
17        x = x + self.mlp(self.ln_2(x))
18        return x
```

The nature of their training

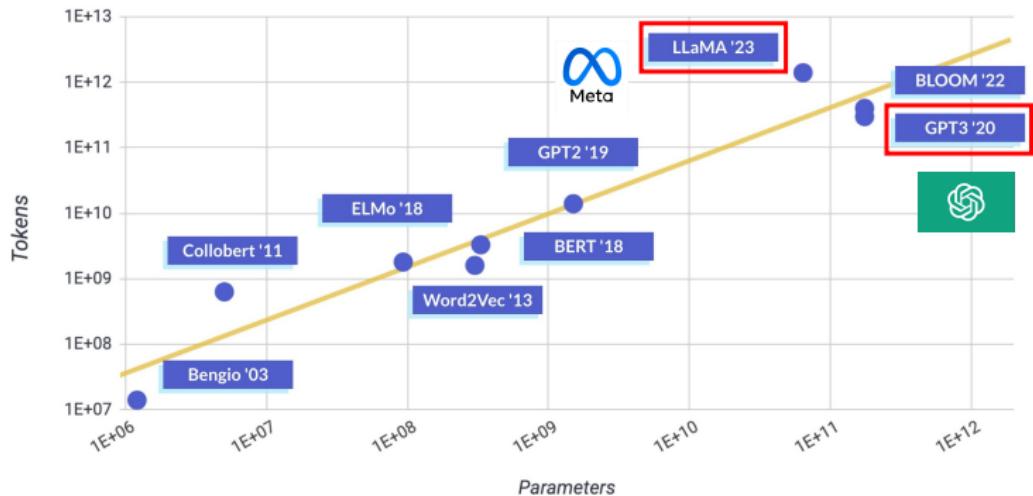
# Model and data scale: a glance



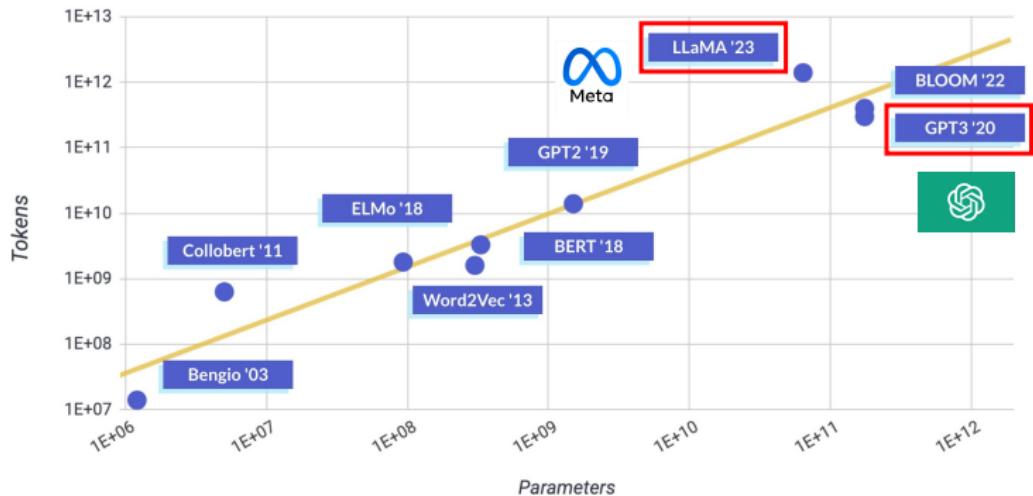
# Model and data scale: a glance



# Model and data scale: a glance



# Model and data scale: a glance

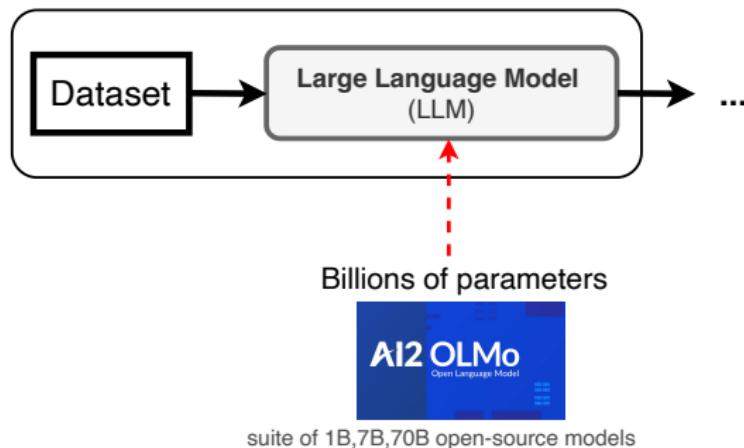


- ▶ Industry is currently dominating, we know increasingly less about what these models are and how they are trained.

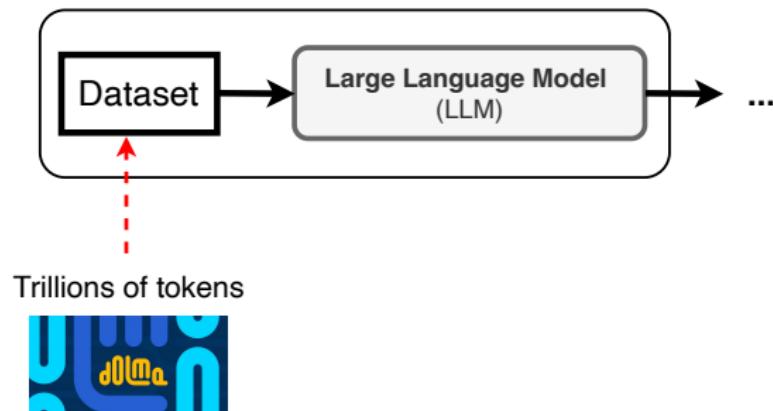
# Open Language Model (OLMo) project at AI2



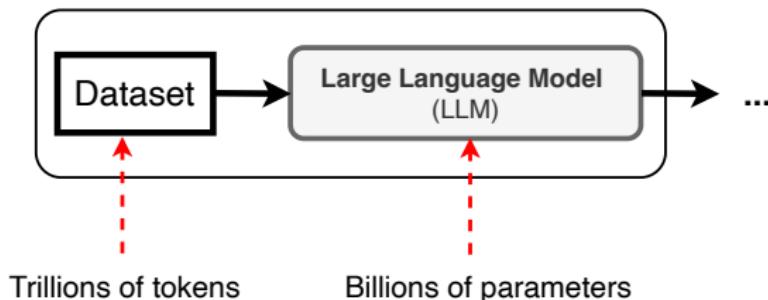
# Open Language Model (OLMo) project at AI2



# Open Language Model (OLMo) project at AI2



# Open Language Model (OLMo) project at AI2



# Where does the data come from?

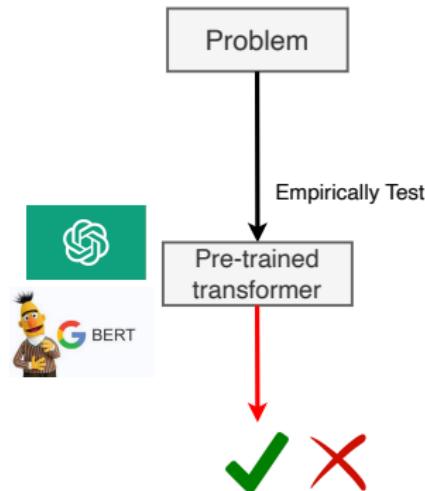
Doc Type	UTF-8 bytes (GB)	Documents (millions)	Unicode words (billions)
web pages	9,022	3,370	1,775
code	1,043	210	260
web pages	790	364	153
social media	339	377	72
STEM papers	268	38.8	50
books	20.4	0.056	4.0
encyclopedic	16.2	6.2	3.7
	<b>11,519</b>	<b>4,367</b>	<b>2,318</b>

What can models do? What do they know?

## What can models do? Different approaches

Problem

# What can models do? Different approaches



# What can models do? Different approaches

**Transformers as Soft Reasoners over Language**

Peter Clark, Oyvind Tafjord, Kyle Richardson  
Allen Institute for AI, Seattle, WA  
[{peterc,oyvindt,kyler}@allenai.org](mailto:{peterc,oyvindt,kyler}@allenai.org)

Can Transformers Reason About Effects of Actions?  
Pratikay Baralaj, Oliver Hinsz, Max Lee, Arman Mirzaei, Kunal Patel,  
Eric C. Stachowiak, Naman Doshi, and Karthik Sridharan  
Arizona State University, Microsoft, New Mexico State University  
[pbaralaj@cs.asu.edu](mailto:pbaralaj@cs.asu.edu), [ohinsz@cs.asu.edu](mailto:ohinsz@cs.asu.edu), [mllee@cs.asu.edu](mailto:mllee@cs.asu.edu), [amir.mirzaei@nmsu.edu](mailto:amir.mirzaei@nmsu.edu), [kpatel@cs.asu.edu](mailto:kpatel@cs.asu.edu), [n.doshi@cs.asu.edu](mailto:n.doshi@cs.asu.edu), [k.sridharan@cs.asu.edu](mailto:k.sridharan@cs.asu.edu)

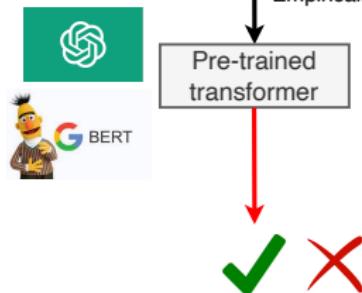
When Can Transformers Count to n?

Gilad Yehudai<sup>1</sup>, Haim Kaplan<sup>2,3</sup>, Asma Ghandehariou<sup>4</sup>, Mor Geva<sup>2,3</sup>, and Amir Globerson<sup>1,3</sup>

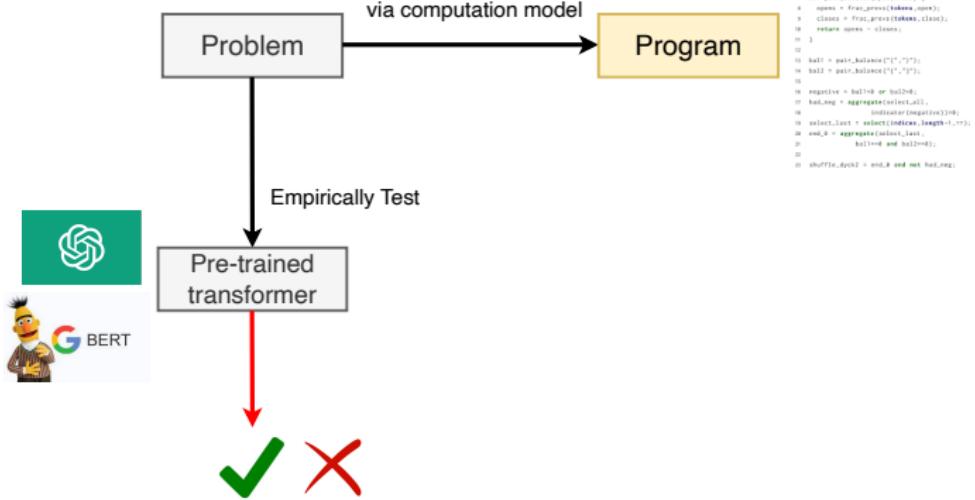
---

Faith and Fate:  
Limits of Transformers on Compositionality

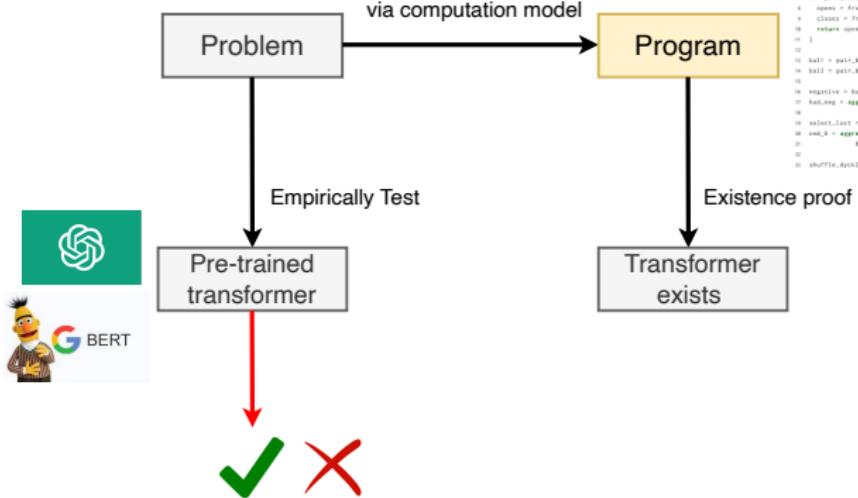
---



# What can models do? Different approaches



# What can models do? Different approaches



```
1 def frac_percent(samp, val):
2     presamp = select(samp, indices, val);
3     return aggregate(presamp,
4                      indicator(samp==val));
5
6
7 def pair_balance(pair, class_):
8     opens = frac_percent(tables, open);
9     classes = frac_percent(tables, class_);
10    return opens - classes;
11
12 half1 = pair_balance(["1","2"]);
13 half2 = pair_balance(["1","3"]);
14 negative = half1 >= half2 &gt;
15 half1.neg = aggregate(select_all,
16                        indicator(pair[1]==0));
17 select_list = tables[tables["id"]<length-1];
18 end1, end2 = aggregate(select_list,
19                        half1 >= half2 &gt;
20                        half1 >= 0);
21
22 shuffle(end1, end2) + end2, neg;
```

# Thinking like a transformer

---

## Thinking Like Transformers

---

Gail Weiss<sup>1</sup> Yoav Goldberg<sup>2,3</sup> Eran Yahav<sup>1</sup>

- ▶ Rasp (*Restricted Access Sequence Processing Language*): a symbolic functional language for expressing transformer computation.

# Thinking like a transformer

```
1  class SimpleBlock(nn.Module):
2      def __init__(self, config):
3          super().__init__()
4          self.attn = SelfAttention(config)
5          self.mlp = MLP(config)
6
7      def forward(self, x):
8          x = x + self.attn(x) #<--- looking around
9          x = x + self.mlp(x) #<--- element-wise
10         #   | residual
11         return x
```

# Thinking like a transformer

```
1  class SimpleBlock(nn.Module):
2      def __init__(self, config):
3          super().__init__()
4          self.attn = SelfAttention(config)
5          self.mlp = MLP(config)
6
7      def forward(self, x):
8          x = x + self.attn(x) #<--- looking around
9          x = x + self.mlp(x) #<--- element-wise
10         #   | residual
11         return x
```

Conditions on information processing

# Thinking like a transformer

```
1 class SimpleBlock(nn.Module):
2     def __init__(self, config):
3         super().__init__()
4         self.attn = SelfAttention(config)
5         self.mlp = MLP(config)
6
7     def forward(self, x):
8         x = x + self.attn(x) #<--- looking around
9         x = x + self.mlp(x) #<--- element-wise
10        # | residual
11        return x
```

Conditions on information processing

- ▶ Each transformation on  $x$  must produce an object of the same size

# Thinking like a transformer

```
1 class SimpleBlock(nn.Module):
2     def __init__(self, config):
3         super().__init__()
4         self.attn = SelfAttention(config)
5         self.mlp = MLP(config)
6
7     def forward(self, x):
8         x = x + self.attn(x) #<--- looking around
9         x = x + self.mlp(x) #<--- element-wise
10        # | residual
11        return x
```

## Conditions on information processing

- ▶ Each transformation on  $x$  must produce an object of the same size
- ▶ Information across positions can only be shared via attention.

# Thinking like a transformer

```
1 class SimpleBlock(nn.Module):
2     def __init__(self, config):
3         super().__init__()
4         self.attn = SelfAttention(config)
5         self.mlp = MLP(config)
6
7     def forward(self, x):
8         x = x + self.attn(x) #<--- looking around
9         x = x + self.mlp(x) #<--- element-wise
10        # | residual
11        return x
```

## Conditions on information processing

- ▶ Each transformation on  $x$  must produce an object of the same size
- ▶ Information across positions can only be shared via attention.
- ▶ Other transformations are element-wise, must have a way to pass around old information.

# Thinking like a transformer

```
1 class SimpleBlock(nn.Module):
2     def __init__(self, config):
3         super().__init__()
4         self.attn = SelfAttention(config)
5         self.mlp = MLP(config)
6
7     def forward(self, x):
8         x = x + self.attn(x) #--- looking around
9         x = self.mlp(x) #--- element-wise
10    return x
```

**Important details omitted!**

## Conditions on information processing

- ▶ Each transformation on  $x$  must produce an object of the same size
- ▶ Information across positions can only be shared via attention.
- ▶ Other transformations are element-wise, must have a way to pass around old information.

# Thinking like a transformer: reversing a string

## Reversing a string

```
text: ["this", "is", "my", "text", "</s>"]
pos: [0, 1, 2, 3, 4]
```

```
reverse  ["<s>", "text", ..., "this"]
```

# Thinking like a transformer: reversing a string

## Reversing a string

```
text: ["this", "is", "my", "text", "</s>"]  
pos: [0, 1, 2, 3, 4]
```

length

[4, 4, 4, 4, 4]



end symbol

# Thinking like a transformer: reversing a string

## Reversing a string

```
text: ["this", "is", "my", "text", "</s>"]  
pos: [0, 1, 2, 3, 4]
```



```
length      [4, 4, 4, 4]  
length - pos [4, 3, 2, 1, 0]
```

# Thinking like a transformer: reversing a string

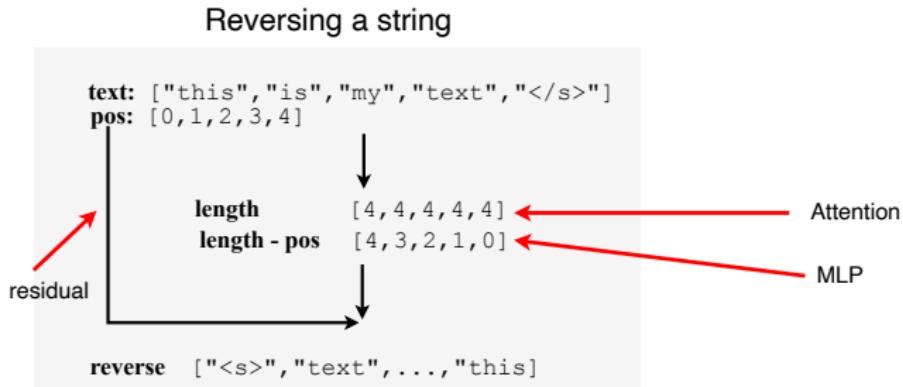
## Reversing a string

```
text: ["this", "is", "my", "text", "</s>"]
pos: [0, 1, 2, 3, 4]
```

length [4, 4, 4, 4, 4]  
length - pos [4, 3, 2, 1, 0]

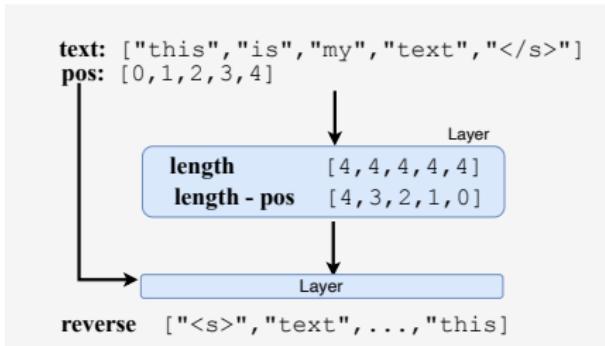
```
reverse ["<s>", "text", ..., "this"]
```

# Thinking like a transformer: reversing a string



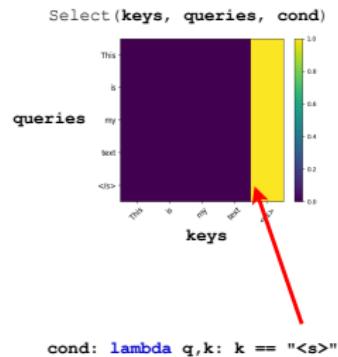
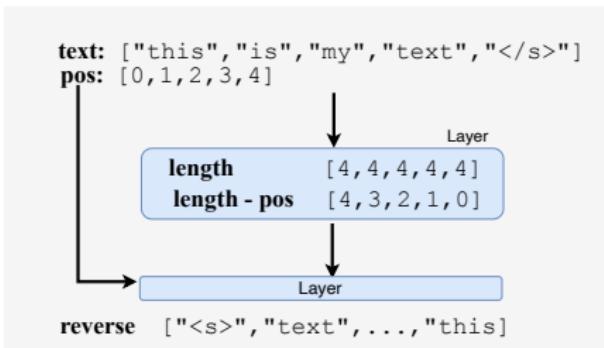
# Thinking like a transformer: reversing a string

## Reversing a string



# Thinking like a transformer: reversing a string

## Reversing a string

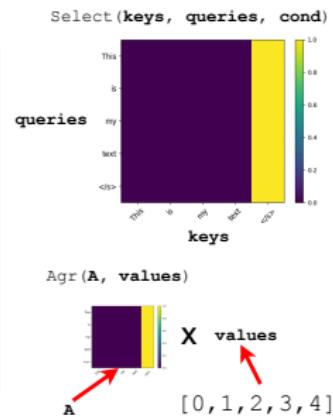
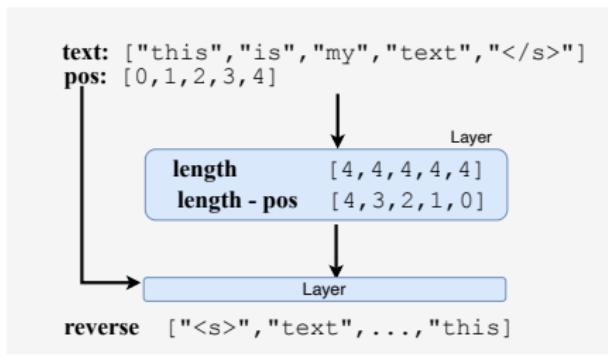


# Thinking like a transformer: reversing a string



# Thinking like a transformer: reversing a string

## Reversing a string



# Thinking like a transformer: reversing a string

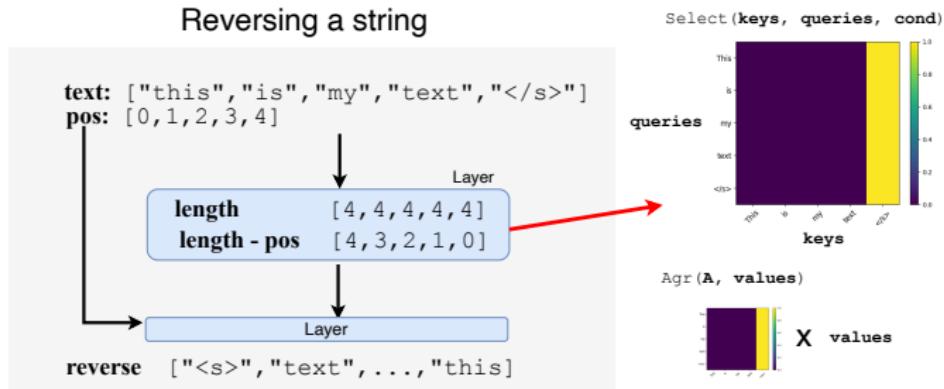
Reversing a string

Select(keys, queries, cond)

**res=aggregate(s, [4,6,8])**

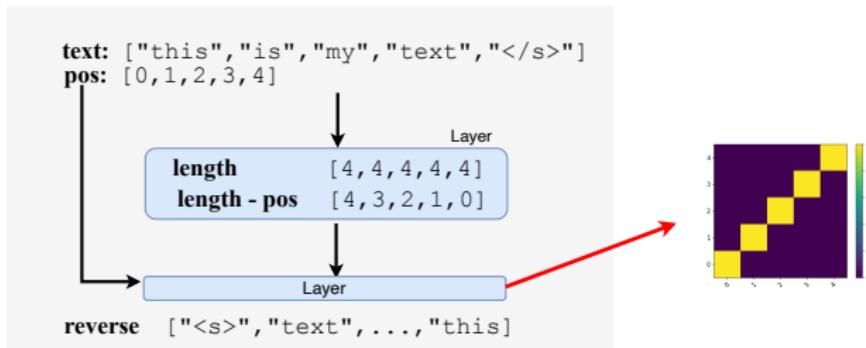
				<b>4 6 8</b>		
F	F	F	4	6	8	=> 0
T	F	F	<b>4</b>	<b>6</b>	<b>8</b>	=> 4 => <b>[0,4,7]</b>
F	T	T	<b>4</b>	<b>6</b>	<b>8</b>	=> 7 ]

# Thinking like a transformer: reversing a string



# Thinking like a transformer: reversing a string

## Reversing a string



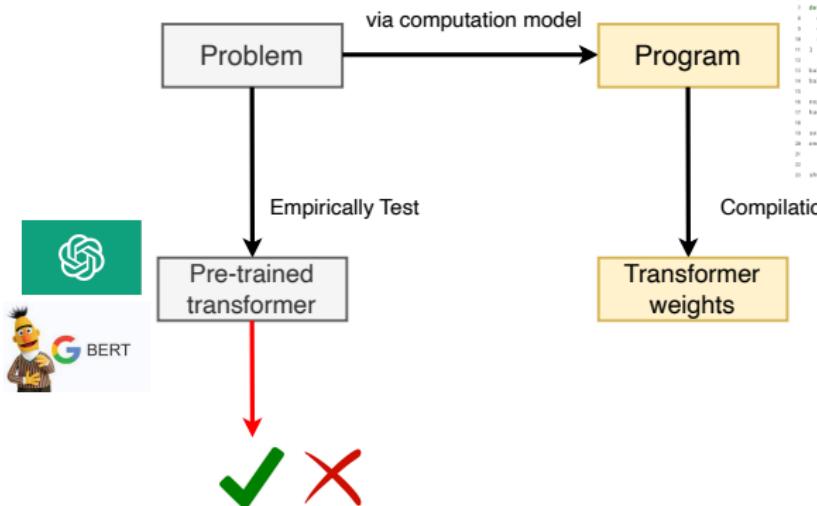
# A small implementation of Rasp

```
1 ## credits: https://github.com/apple/ml-np-rasp
2 import numpy as np
3
4 def indices(x):
5     return np.arange(len(x), dtype=int)
6 def select(k,q,pred):
7     s = len(k)
8     A = np.zeros((s, s), dtype=bool)
9     for i in range(s):
10         for j in range(s): A[i,j] = pred(q[i],k[j])
11     return A
12 def sel_width(A):
13     return np.dot(A, np.ones(len(A))).astype(int)
14 def aggr(A, v):
15     out = np.dot(A, v)
16     norm = sel_width(A)
17     out = np.divide(out,norm,where=(norm != 0))
18     return out.astype(int)
```

# A small implementation of Rasp

```
1 pos = indices(tokens)
2 EOS = 0
3
4 ##### layer 1
5 # attn layer
6 length = aggr(
7     select(tokens,tokens,lambda q,k : k == EOS),
8     pos
9 )
10 targets = length - pos #<--- MLP
11
12 ##### layer 2
13 output = aggr(
14     select(pos,targets,lambda k,v: k == v),
15     tokens
16 )
```

# The bigger picture now



```
1 def frac_percent(samp, val):
2     pres = select(lambda i: samp[i].val == val);
3     return aggregate(pres, lambda i: indicator(samp==val));
4 
5 
6 def pair_balance(pair, class_):
7     opens = frac_percent(pair.takes, class_);
8     classes = frac_percent(pair.takes, class_);
9     return opens - classes;
10 
11 
12 half = pair_balance(["1","2"]);
13 half = pair_balance(["1","3"]);
14 negative = half>0 or half<0;
15 badLang = aggregate(select(lambda i: i.lang != "en", pair.takes));
16 selectLang = select(lambda i: i.lang == "en", pair.takes);
17 endLang = aggregate(select_lang, lambda i: half>0 and i.lang=="en");
18 endLang = aggregate(select_lang, lambda i: half<0 and i.lang=="en");
19 
20 shuffle(half1 + endLang + badLang);
```

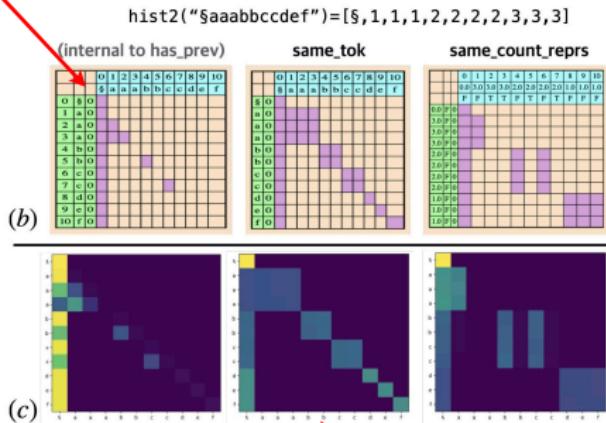


# Does a transformer implement my algorithm?

```
1 same_tok = select(tokens,tokens,==);
2 hist = selector_width(
3     same_tok,
4     assume_bos = True);
5
6 first = not has_prev(tokens);
7 same_count = select(hist,hist,==);
8 same_count_reprs = same_count and
9     select(first,True,==);
10
11 hist2 = selector_width(
12     same_count_reprs,
13     assume_bos = True);
```

Rasp program

Target attention



Language	Layers	Heads	Test Acc.	Attn. Matches?
Reverse	2	1	99.99%	✓
Hist BOS	1	1	100%	✓
Hist no BOS	1	2	99.97%	✓
Double Hist	2	2	99.58%	✓
Sort	2	1	99.96%	✗
Most Freq	3	2	95.99%	✗
Dyck-1 PTF	2	1	99.67%	✓
Dyck-2 PTF <sup>8</sup>	3	1	99.85%	✗

## Tracr: Compiled Transformers as a Laboratory for Interpretability

---

David Lindner<sup>†</sup>  
Google DeepMind

János Kramář  
Google DeepMind

Sebastian Farquhar  
Google DeepMind

Matthew Rahtz  
Google DeepMind

Thomas McGrath  
Google DeepMind

Vladimir Mikulik<sup>†</sup>  
Google DeepMind

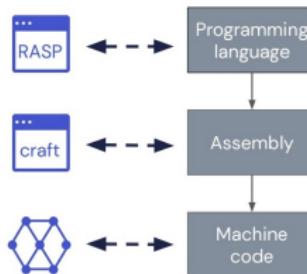


Figure 3: Tracr translates RASP to Craft and then to model weights, analogous to how programming languages are first translated to assembly then to machine code.

## Learning Transformer Programs

Dan Friedman Alexander Wettig Danqi Chen

Department of Computer Science & Princeton Language and Intelligence  
Princeton University

{dfriedman,awettig,danqic}@cs.princeton.edu

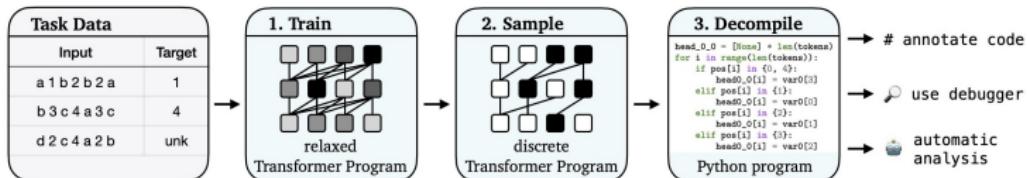
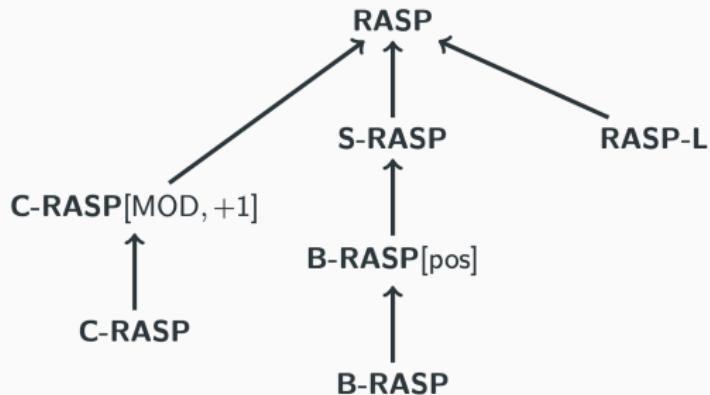


Figure 1: We design a modified Transformer that can be trained on data and then automatically discretized and converted into a human-readable program. The program is functionally identical to the Transformer, but easier to understand—for example, using an off-the-shelf Python debugger.

# Different variants of Rasp



Disclaimer: More arrows may exist

Taken from Andy Lang [https://pentagonalize.github.io/files/CLAY\\_slides.pdf](https://pentagonalize.github.io/files/CLAY_slides.pdf)

# Conclusions

- ▶ **covered today:** Language modeling basics, transformer architecture and model training.

# Conclusions

- ▶ **covered today:** Language modeling basics, transformer architecture and model training.
- ▶ Rasp: symbolic programming language for describing computation in transformers.

# Conclusions

- ▶ **covered today:** Language modeling basics, transformer architecture and model training.
- ▶ Rasp: symbolic programming language for describing computation in transformers.
  - Useful tool for interpretability and proving properties of transformer abilities.

# Conclusions

- ▶ **covered today:** Language modeling basics, transformer architecture and model training.
- ▶ Rasp: symbolic programming language for describing computation in transformers.
  - Useful tool for interpretability and proving properties of transformer abilities.

**Next lecture:** Declarative approaches to model training.

Thank you.

## References I

- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.