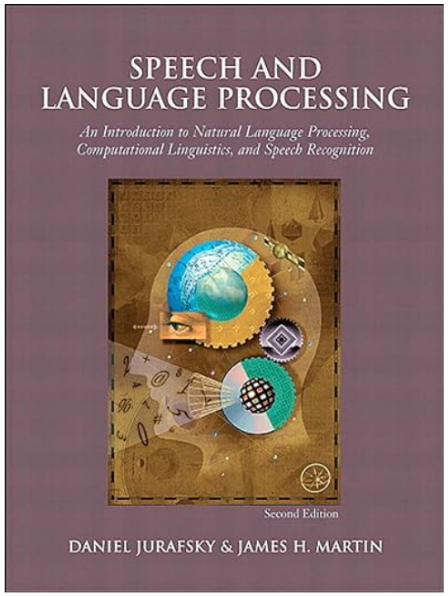
# Transformery (parametry, warianty atencji, trenowanie)

Paweł Rychlikowski

Instytut Informatyki UWr

7.01.2026

# Literatura do przedmiotu




- Najbardziej wszechstronną książką o NLP jest Speech and Language Processing D.Jurafsky i H.Martin (hura!)
- ... ale jej ostatnie (drugie) wydanie to rok 2011 (łeee)
- ... ale autorzy pracują nad kolejnym (hura!)
- ... ale na pytanie, kiedy skończą odpowiadają na oficjalnej stronie Don't ask. (łeee)
- ... ale dopóki nie skończą, na stronie są pdf-y z aktualnymi wersjami rozdziałów (z nich brana jest część obrazków do tego wykładu)

# Treść książki

Link: `https://web.stanford.edu/~jurafsky/slp3/`
(ostatni update był **wczoraj**!)

## Wiedza wstępna

- R4: Logistic Regression
- R7: Neural networks

## Ważna część naszego wykładu

- R6: Vector Semantics and Embeddings
- R9: Transformers
- R10: Large Language Models
- R11: Masked Language Models
- R12: Model Alignment, Prompting, and In-Context Learning

# Treść książki

## Treści uzupełniające

- R8: RNNs and LSTMs
- R13: Machine Translation
- R14: Question Answering, Information Retrieval, and RAG
- R15: Chatbots and Dialogue Systems
- R16: Automatic Speech Recognition and Text-to-Speech
- R17: Sequence Labeling for Parts of Speech and Named Entities

# Treść książki

## Cmentarzyk

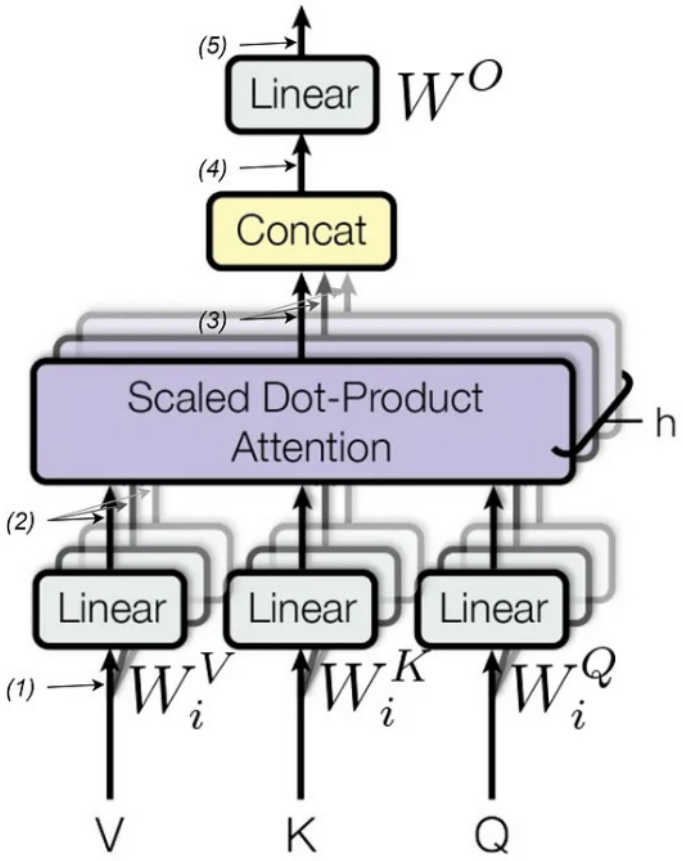
Rozdziały, które nie będą włączone do książki (jedynie w sieci)

- A: Hidden Markov Models
- B: Spelling Correction and the Noisy Channel
- C: Statistical Constituency Parsing
- D: Context-Free Grammars
- E: Combinatory Categorial Grammar
- F: Logical Representations of Sentence Meaning
- G: Word Senses and WordNet
- H: Phonetics

# Literatura do przedmiotu (cd)

- Bardziej zorientowaną na programistów książką jest **Przetwarzanie języka naturalnego z wykorzystaniem transformerów**, Tunstral, von Werra, Wolf (Helion, O'Reily)
- I jest wiele innych książek, na przykład Helion ma pewnie kilkadziesiąt pozycji dotyczących LM/LLM

# Liczba parametrów transformera (atencja)




$$W^O.shape = (d_{model}, d_{model})$$
$$W_i^Q.shape = (d_{model}, d_{qkv})$$
$$W_i^K.shape = (d_{model}, d_{qkv})$$
$$W_i^V.shape = (d_{model}, d_{qkv})$$

$$d_{qkv} = d_k = d_v = d_{model}/n_{heads} = d_{model}/h$$

$1)(b, d_{model})$
$2)(b, d_{qkv})$
$3)(b, d_{qkv})$
$4)(b, d_{model})$
$5)(b, d_{model})$

Transformer multi-head attention. Adapted from figure 2 from the public domain paper

- Wektory przetwarzane przez transformer grupowane są we wsadach (batch), liczba wsadów to $b$
- (...)

# Liczba parametrów transformera (atencja)

## Obliczenia liczby parametrów

$$N_{attention} = \overbrace{(d_{model} * d_{model} + d_{model})}^{W^O} + \overbrace{(d_{model} * d_{qkv} + d_{qkv})}^{W_i^Q, W_i^K, or\ W_i^V} * \overbrace{n_{heads}}^{\text{Three matrices for every head}} * 3 =$$

$$= (d_{model} * d_{model} + d_{model}) + (d_{model} * d_{model} + d_{model}) * 3 =$$

$$(d_{model} * d_{model} + d_{model}) * 4 = \underbrace{(d_{model}^2 + d_{model}) * 4}_{\text{The exact formula}} \approx \underbrace{4 * d_{model}^2}_{\text{The approximate formula}}$$

---

$$Why\ (d_{model} * d_{qkv} + d_{qkv}) * n_{heads} = (d_{model} * d_{model} + d_{model}):$$

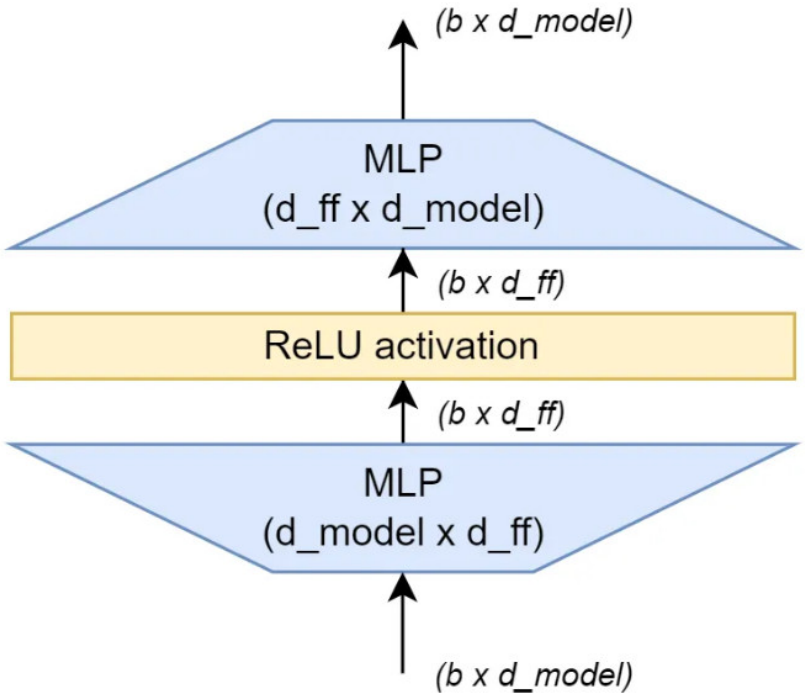
$$(d_{model} * d_{qkv} + d_{qkv}) * n_{heads} =$$

$$d_{model} * \underbrace{d_{qkv} * n_{heads}}_{d_{model},\ since\ d_{qvk} = d_{model}/n_{heads}} + \underbrace{d_{qkv} * n_{heads}}_{d_{model}} = d_{model} * d_{model} + d_{model}$$

The formula for calculating the number of parameters in the Transformer attention module. Image by Author

Źródło: `https://towardsdatascience.com/how-to-estimate-the-number-of-parameters-in-transformer-models-ca0f`

# Liczba parametrów transformera (feedforward)




Transformer feed-forward network. Image by Author

$$N_{feedforward} = \overbrace{(d_{model} * d_{ff} + d_{ff})}^{\text{The first linear layer}} + \overbrace{(d_{ff} * d_{model} + d_{model})}^{\text{The second linear layer}} =$$
$$d_{model} * d_{ff} + d_{ff} * d_{model} + d_{model} + d_{ff} =$$
$$\underbrace{2 * d_{model} * d_{ff} + d_{model} + d_{ff}}_{\text{The exact formula}} \approx \underbrace{2 * d_{model} * d_{ff}}_{\text{The approximate formula}}$$

The formula for calculating the number of parameters in the Transformer feed-forward net. Image by Author

# Liczba parametrów transformera (feedforward)

- LayerNormalization ma $2d$ parametrów
- Jeżeli w słowniku mamy V elementów, to dochodzi to tego $d \times V$ osadzeń słów
- Parametry sieci FF i atencji dotyczą jednej warstwy, trzeba je zatem pomnożyć przez liczbę warstw.

## Ostateczny wzór (w przybliżeniu)

$$12d^2 \times L + |V| \times d$$

gdy $d_{\text{ff}} = 4d$

# Liczba parametrów transformera

## Ostateczny wzór (w przybliżeniu)

$$12d^2 \times L + |V| \times d$$

gdy $d_{\text{ff}} = 4d$

- Wielkości (b.orientacyjnie):
  - $d$: 500-10000
  - $L$: 10-50
  - $|V|$: 50K-300K
- W typowym modelu $12dL > |V|$

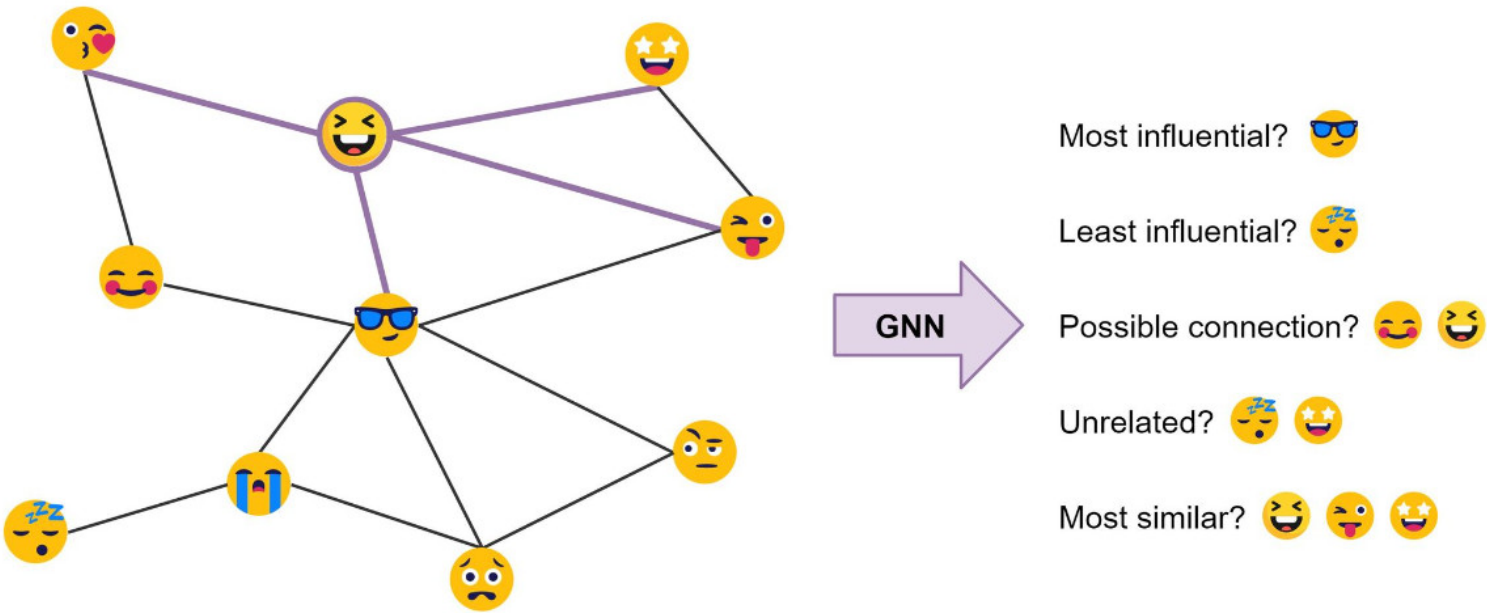
# Anatomia papugi

Zobaczmy, jak wygląda model papugi (w notatniku)

# Transformery to sieci grafowe

## Definicja

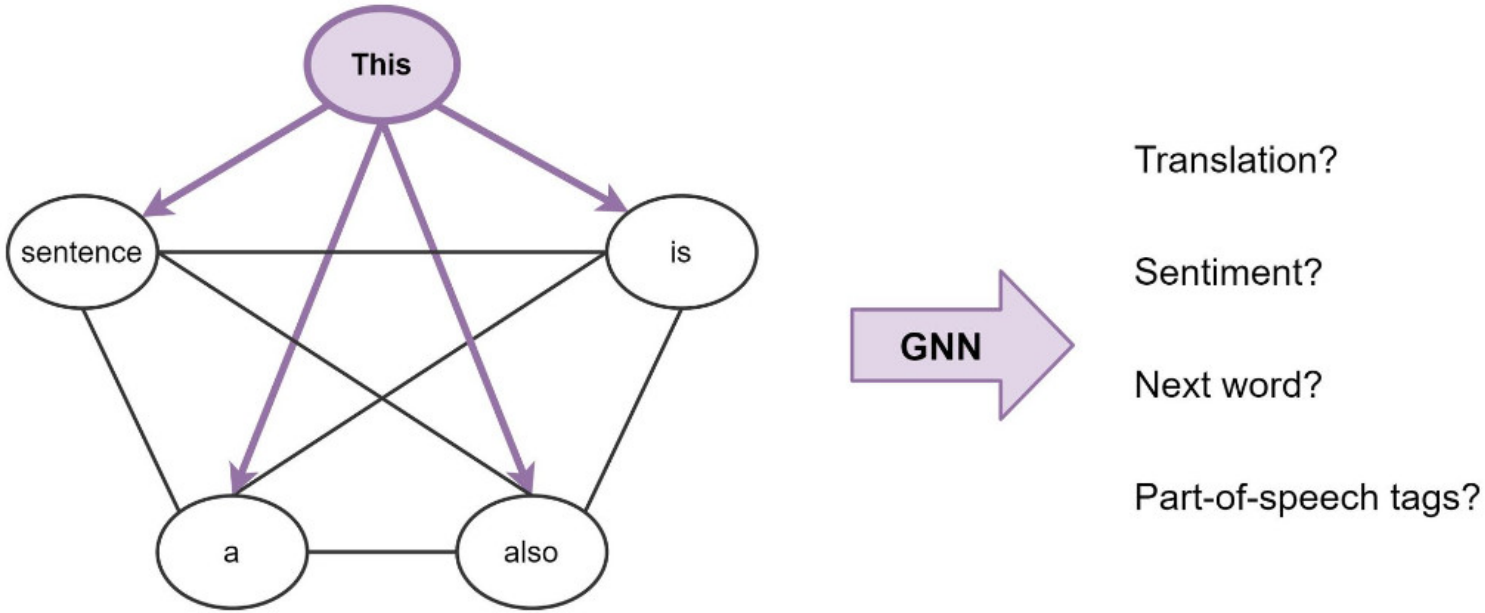
**Neuronowe sieci grafowe (GNN)** to sieci, dla których daną wejściową jest graf (na ogół skierowany)




Take the example of this emoji social network: The node features produced by the GNN can be used for predictive tasks such as identifying the most influential members or proposing potential connections.

Źródło:
`https://thegradient.pub/transformers-are-graph-neural-networks/`

# Transformery to sieci grafowe




## Uwaga

Zdanie możemy widzieć jako graf w którym węzłami są słowa/tokeny

# Transformery to sieci grafowe

$$\mathbf{A} = \text{softmax}\left(\text{mask}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\right)\mathbf{V} \tag{11.1}$$

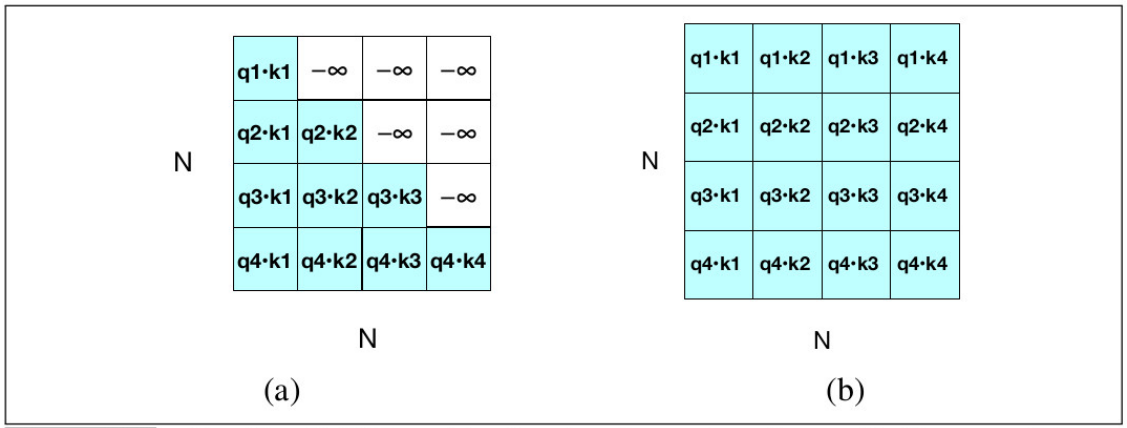
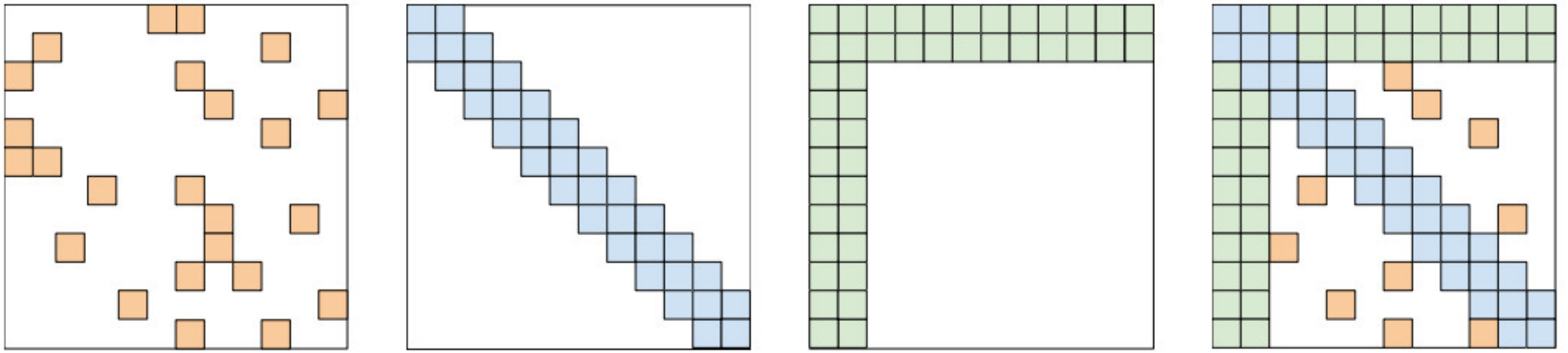



**Figure 11.2** The $N \times N$ $\mathbf{QK}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

## Uwaga

- Możemy utożsamiać tokeny z węzłami w grafie.
- Za pomocą odpowiedniej maski możemy sprawić, żeby atencja przebiegała tylko pomiędzy sąsiednimi węzłami w grafie.
- Maska atencji $\approx$ macierz incydencji

# Inne schematy atencji w transformerach dla sekwencji



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

- Atencja losowa nawiązuje do idei **małego świata** (każdy jest 6 hand-szejków od np. Donalda Trumpa)
- Atencja globalna wprowadza specjalne węzły, które zbierają informację (i tylko z nimi się inne węzły łączą)
- Atencja w oknie, w której każdy węzeł łaczy się z bliskimi sąsiadami, przypomina trochę sieci konwolucyjne (CNN)

# BigBird – konkluzje

- BIGBIRD satisfies a number of theoretical results: it is a universal approximator of sequence to sequence functions and is also Turing complete.
- Empirically, BIGBIRD gives state-of-the-art performance on a number of NLP tasks such as question answering and long document classification.

Big Bird: Transformers for Longer Sequences, 2021

# Transformery wnioskujące. Brudnopisy

## SHOW YOUR WORK: SCRATCHPADS FOR INTERMEDIATE COMPUTATION WITH LANGUAGE MODELS

**Maxwell Nye**[12*] **Anders Johan Andreassen**[3] **Guy Gur-Ari**[3] **Henryk Michalewski**[2]

**Jacob Austin**[2] **David Bieber**[2] **David Dohan**[2] **Aitor Lewkowycz**[3] **Maarten Bosma**[2]

**David Luan**[2] **Charles Sutton**[2] **Augustus Odena**[2]

[1]MIT
[2]Google Research, Brain Team
[3]Google Research, Blueshift Team

### ABSTRACT

Large pre-trained language models perform remarkably well on tasks that can be done "in one pass", such as generating realistic text (Brown et al., 2020) or synthesizing computer programs (Chen et al., 2021; Austin et al., 2021). However, they struggle with tasks that require unbounded multi-step computation, such as adding integers (Brown et al., 2020) or *executing* programs (Austin et al., 2021). Surprisingly, we find that these same models are able to perform complex multi-step computations—even in the few-shot regime—when asked to perform the operation "step by step", showing the results of intermediate computations. In particular, we train Transformers to perform multi-step computations by asking them to emit intermediate computation steps into a "scratchpad". On a series of increasingly complex tasks ranging from long addition to the execution of arbitrary

# Transformery wnioskujące. Brudnopisy

```
Input:
2 9 + 5 7

Target:
<scratch>
2 9 + 5 7 ,  C: 0
2 + 5 , 6 C: 1  # added 9 + 7 = 6 carry 1
, 8 6 C: 0  # added 2 + 5 + 1 = 8 carry 0
0 8 6
</scratch>
8 6
```

Figure 2: Example of input and target for addition with a scratchpad. The carry is recorded in the digit following "C:". Comments (marked by #) are added for clarity and are not part of the target.

Dodawanie możemy rozpisać cyfra po cyfrze, z przeniesieniem.

# Transformery z brudnopisem. Skuteczność dodawania

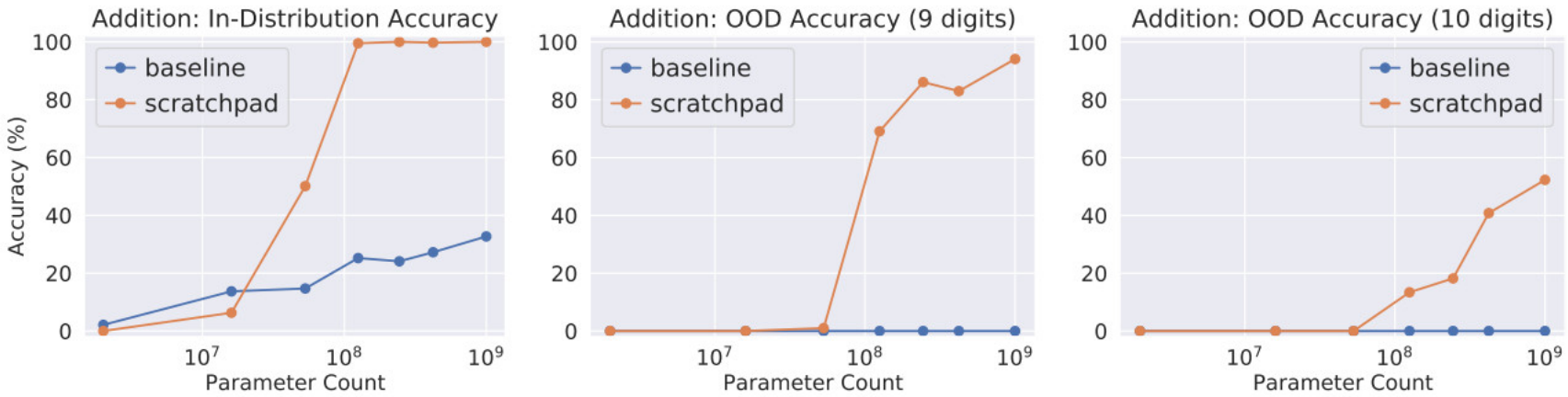



Figure 3: Using a scratchpad significantly improves the performance of pre-trained Transformer-based models on addition, including their ability to generalize out of the training distribution to numbers with more digits. Models were trained on 1-8 digit addition. The baseline models were trained without intermediate scratchpad steps.

Używane były tu modele wstępnie wytrenowane na stronach WWW (korpusach „ogólnych"), które następnie dotrenowywano na danych z brudnopisem lub bez.

Trening „from-scratch" z odpowiednią tokenizacją dałoby się zrobić na mniejszych modelach).

# Transformery z brudnopisem. Ewaluacja wielomianów

```
Input:
Evaluate -7*x**2 + 7*x + 5 at x = 1

Target:
<scratch>
-7*x**2: -7
7*x: 7
5: 5
</scratch>
total: 5
```

Figure 4: Example of polynomial evaluation with a scratchpad. Each term in the polynomial is computed separately and then added.

Table 1: Results for polynomial evaluation task. Scratchpad outperforms direct prediction whether using fine-tuning or few-shot.

| | Few-shot | Fine-tuning |
|---|---|---|
| Direct prediction | 8.8% | 31.8% |
| Scratchpad | **20.1%** | **50.7%** |

# Transformery z brudnopisem. Wykonywanie programów

**DIRECT EXECUTION PREDICTION**

```
Consider the following Python function:

def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s

Fill in the ??? below:
assert remove_Occ("PHP","P") == ???
```

→ Large Language Model →

```
assert remove_Oct("PHP", "P") == "H"
```

**SCRATCHPAD TRACING**

```
Consider the following Python function:

def remove_Occ(s,ch):
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    for i in range(len(s) - 1,-1,-1):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
    return s

output = remove_Occ("PHP","P")

What is the execution trace?
```

→ Large Language Model →

```
state: {}
line: def remove_Occ(s,ch):
state: {"remove_Occ": "<callable_object remove_Occ>"}
line: output = remove_Occ("PHP","P")
state: {"ch": "P", "s": "PHP"}
line:     for i in range(len(s)):
state: {"ch": "P", "s": "PHP", "i": 0}
line:         if (s[i] == ch):
state: {"ch": "P", "s": "PHP", "i": 0}
line:             s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "HP", "i": 0}
line:             break
state: {"ch": "P", "s": "HP", "i": 0}
line:     for i in range(len(s) - 1,-1,-1):
state: {"ch": "P", "s": "HP", "i": 1}
line:         if (s[i] == ch):
state: {"ch": "P", "s": "HP", "i": 1}
line:             s = s[0 : i] + s[i + 1:]
state: {"ch": "P", "s": "H", "i": 1}
line:             break
state: {"ch": "P", "s": "H", "i": 1}
line:     return s
state: {"remove_Occ": "<callable_object remove_Occ>",
 "output": "H"}
```

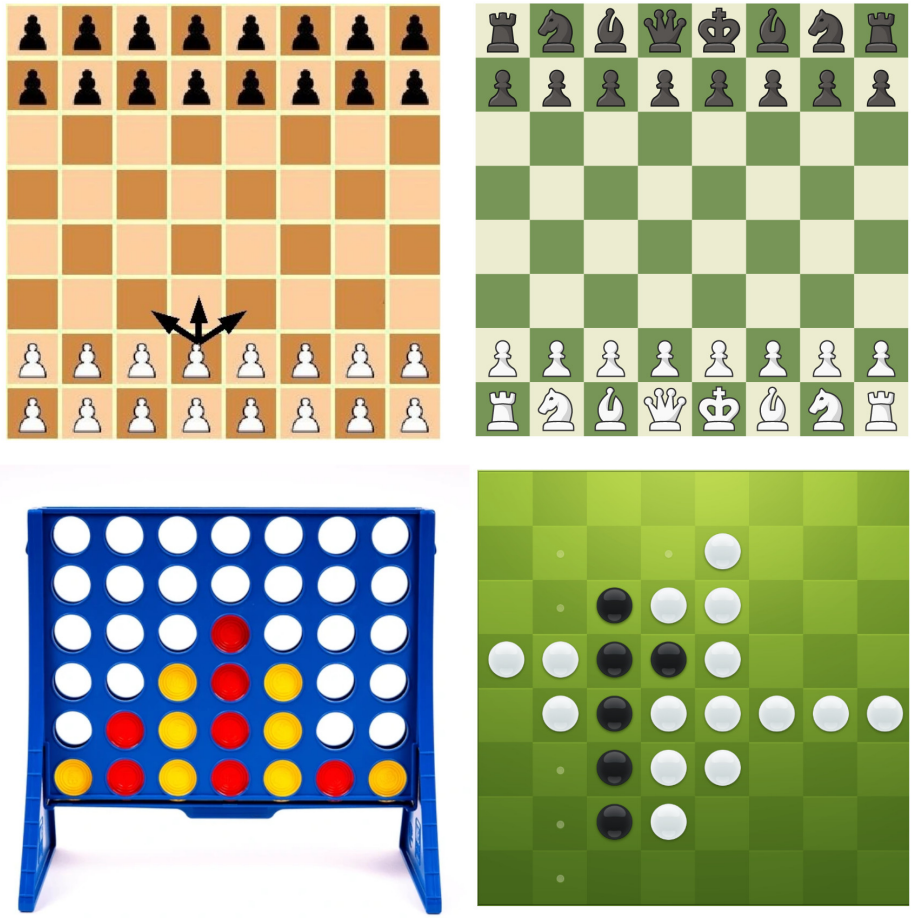
# Czy modele językowe umieją dodawać. Odcinek 2

Popatrzymy na starsze repozytorium Karpathy-ego

- **miniGPT** (Link: `https://github.com/karpathy/minGPT`)
- W oryginalnym `demo.ipynb` było zadanie sortowania
- My popatrzymy na zadanie dodawania liczb trzycyfrowych

Demonstracja `demo_add.ipynb`.

# Gry planszowe



Jeżeli dla ciągów ruchów umiemy przewidywać kolejny ruch i to tym samym umiemy grać w daną grę (okazuje się, że całkiem dobrze).

# Czy autoregresywny model językowy może grać sensownie w szachy?

## Może?

- Może się nauczyć otwarć (lepiej niż ludzie)
- Nawet bardzo proste modele mogą grać lepiej niż losowo (również poza otwarciem)
- (semi)-unigramowy model $P(\text{move}|n)$, gdzie $n$ is numerem ruchu może grać lepiej niż losowy (bo?)

# OthelloGPT

Published as a conference paper at ICLR 2023

## EMERGENT WORLD REPRESENTATIONS: EXPLORING A SEQUENCE MODEL TRAINED ON A SYNTHETIC TASK

**Kenneth Li*** Harvard University
**Aspen K. Hopkins** Massachusetts Institute of Technology
**David Bau** Northeastern University
**Fernanda Viégas** Harvard University
**Hanspeter Pfister** Harvard University
**Martin Wattenberg** Harvard University

### ABSTRACT

Language models show a surprising range of capabilities, but the source of their apparent competence is unclear. Do these networks just memorize a collection of surface statistics, or do they rely on internal representations of the process that generates the sequences they see? We investigate this question in a synthetic setting by applying a variant of the GPT model to the task of predicting legal moves in a simple board game, Othello. Although the network has no a priori knowledge of the game or its rules, we uncover evidence of an emergent nonlinear

# OthelloGPT

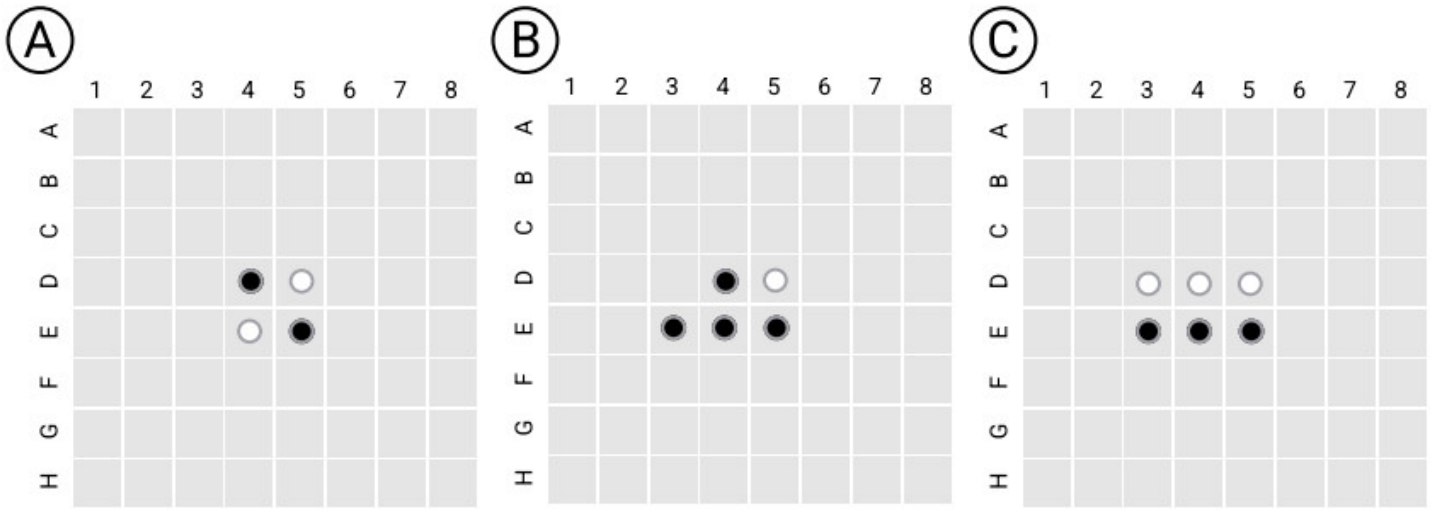



Figure 1: A visual explanation of Othello rules, from left to right: (A) The board is always initialized with four discs (two black, two white) placed in the center of the board. (B) Black always moves first. Every move must flip one or more opponent discs by outflanking—or sandwiching—the opponent disc(s). (C) The opponent repeats this process. A game ends when there are no more legal moves.

Popatrzmy na demo z przedmiotu SI (i zauważmy, jak dynamiczna jest plansza)