MY LITTLE LLM

kurs · Slayer Labs

Wytrenuj własny model językowy od zera

100% from scratch, po polsku — i łagodne wejście w naukę o LLM

Skryptum kursu wideo

wersja jednolita

Wszystkie liczby i wykresy to prawdziwe pomiary z modelu GPT-2 124M trenowanego od zera.

Spis treści

Wstęp

To skryptum towarzyszy kursowi wideo **My Little LLM**. Budujemy prawdziwy model GPT-2 124M trenowany od zera na polskim — własny tokenizer, własne dane, własny trening, uczciwa ewaluacja. Bez fine-tuningu cudzego modelu, bez API.

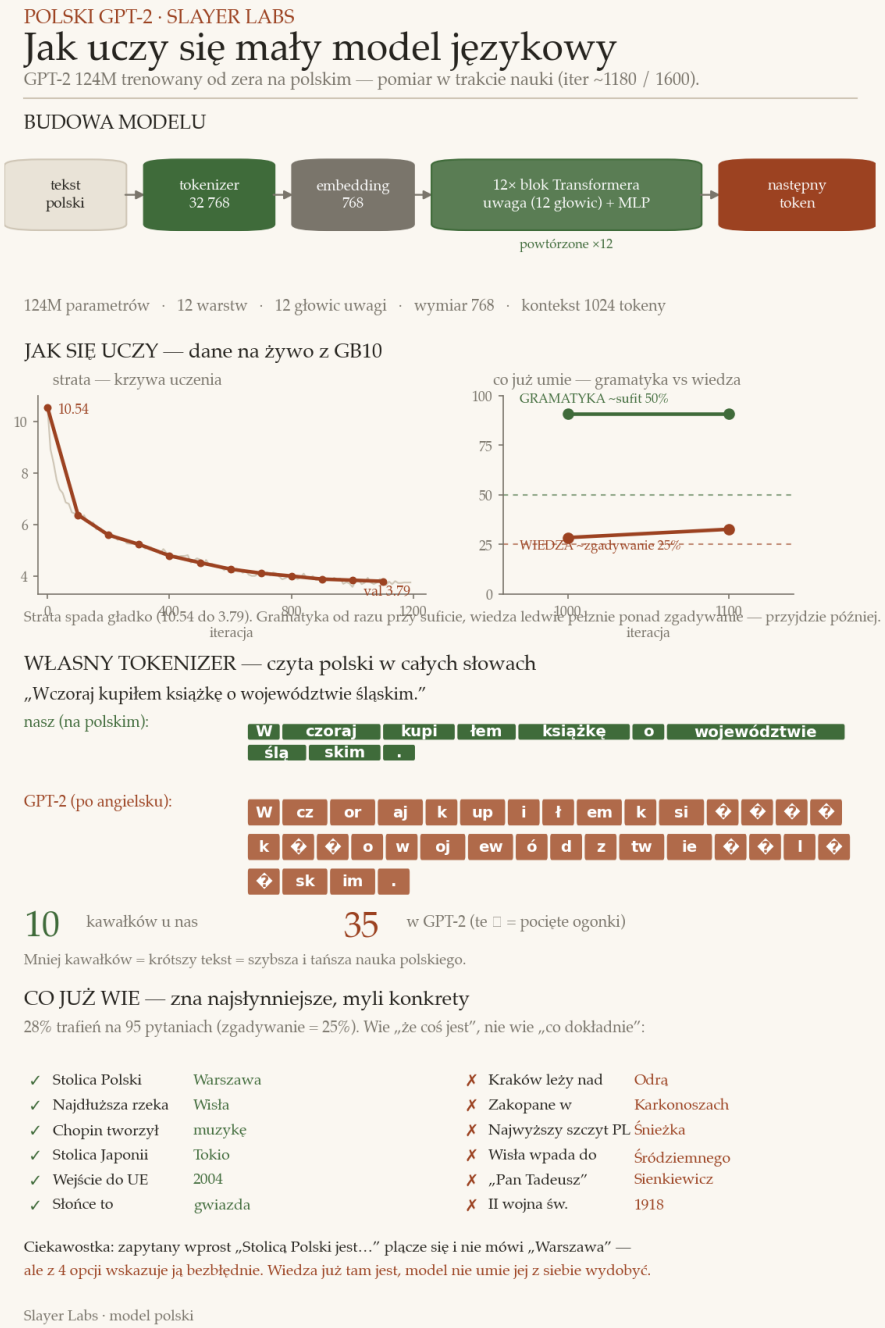
To też **łagodna rampa do prawdziwej nauki o LLM**: każde zjawisko, które zobaczysz na własnym modelu — prawa skalowania, sufit wiedzy, składnia przed semantyką — jest udokumentowanym wynikiem w literaturze. Twój mały model jest mikroskopem do oglądania tego, co dzieje się w wielkich modelach.

Uwaga: zachowano oznaczenia produkcyjne ze scenariuszy wideo — „(wizual: …)" to wskazówka, co pokazać na ekranie w danym momencie.

1 Odcinek 1 — Po co własny model (i kiedy NIE)

Czas: ~10 min **Cel:** Widz rozumie, czym jest trening od zera, jak różni się od fine-tuningu i API, i kiedy to ma sens. Wychodzi z odcinka chcąc zbudować własny model — i wiedząc, że to realne.

1.1 HAK (0:00–0:20)

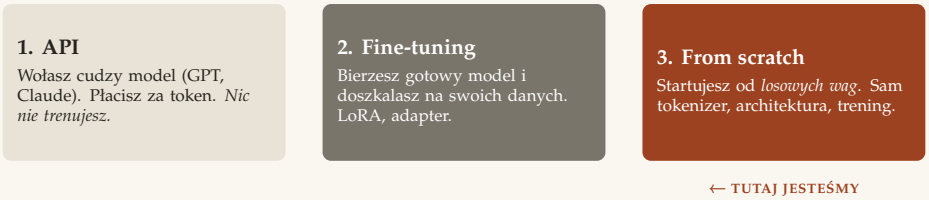


To napisał model, który zbudowałem od zera. Nie fine-tuning ChatGPT. Nie API. Od. Zera.

Własny tokenizer, własne dane, własny trening. 124 miliony parametrów, uczone od losowych wag do czegoś, co pisze poprawną polszczyzną. W tym kursie zrobisz to samo.

1.2 INTRO — trzy poziomy "robienia AI" (0:20–2:00)

Zanim cokolwiek odpalimy, jedno rozróżnienie, które oszczędzi ci miesięcy. Są trzy zupełnie różne rzeczy, które ludzie nazywają "robieniem własnego AI":



99% „własnych modeli AI" w sieci to poziom 1 albo 2. My robimy 3.

1. **API** — wołasz cudzy model (GPT, Claude). Płacisz za token. Nic nie trenujesz.
2. **Fine-tuning** — bierzesz gotowy model i doszkalasz go na swoich danych. Adapter, LoRA. Tanie, szybkie.
3. **From scratch** — startujesz od **losowych wag**. Sam tokenizer, sama architektura, sam trening. To robimy tutaj.

99% "własnych modeli AI" w internecie to poziom 1 albo 2. My robimy 3 — bo dopiero wtedy naprawdę rozumiesz, co siedzi w środku.

1.3 SEKCJA 1 — Po co w ogóle trenować od zera? (2:00–4:30)

Szczerze: w produkcji **najczęściej NIE powinieneś**. Fine-tuning gotowego modelu jest tańszy i lepszy. Więc po co ten kurs?

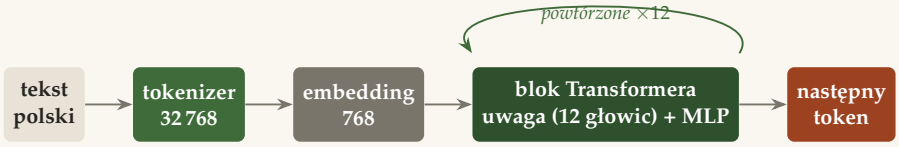
Bo "od zera" to jedyny sposób, żeby naprawdę zrozumieć, jak te modele działają.

(wizual: terminal, `ls` na repo — tokenizer, dane, train.py, eval)

Kiedy przejdziesz cały pipeline raz, przestają być magią: - zrozumiesz, **dlaczego** model halucynuje, - zrozumiesz, **co** robi tokenizer i czemu to wąskie gardło dla polskiego, - zrozumiesz, **kiedy** większy model jest konieczny, a kiedy to przepalanie kasy.

A są też realne powody produkcyjne, dla których ktoś trenuje od zera: - **język/domena** słabo pokryte przez duże modele (np. dobry polski, prawo, medycyna), - **suwerenność** — model na własnym sprzęcie, dane nie wychodzą, - **rozmiar** — malutki model na edge, którego nie da się dostać fine-tuningiem.

1.4 SEKCJA 2 — Co konkretnie zbudujemy (4:30–7:00)



124 M parametrów • 12 warstw • 12 głowic uwagi • wymiar 768 • kontekst 1024 tokeny

GPT-2. Klasyk, ale wystarczająco prawdziwy: ta sama architektura, co duże modele, tylko mniejsza. **124 miliony parametrów. 12 warstw. 12 głowic uwagi.**

Trenujemy go na polskim — i tu pierwszy konkret, który zobaczysz w odcinku 3:

GPT-2 nie zna polskich liter. Tnie "książkę" na 9 kawałków bajtów. Nasz tokenizer od zera robi z tego 1 token. To nie kosmetyka — to 2.8× krótszy tekst, czyli tańszy trening.

I efekt końcowy, do którego dążymy:

POLSKI GPT-2 124M · SLAYER LABS

Co model naprawdę pisze

Prawdziwe próbki z finalnego modelu (iter 1600). Sampling temp 0.6 · top_k 50 · top_p 0.9.

Stolicą Polski jest…

Stolicą Polski jest Polska Polska, która w 2015 roku została wybrana do konkursu (…) Kosmos – miasto w południowej Polsce, w województwie dolnośląskim, w powiecie wrocławskim, leży na prawym brzegu Odry, przy drodze krajowej nr 9.

✗ ZMYŚLONE FAKTY „Polska Polska", wymyślone miasto „Kosmos", nad Odrą zamiast Wisły — ale forma hasła Wikipedii

Polska jest…

Polska jest w stanie Kalifornia, w hrabstwie Liverpool. (…) Hrabstwo Boliwii liczy 3 410 mieszkańców. Hrabstwo Boliwii liczy 5 630 mieszkańców. Hrabstwo Boliwii liczy 9 770 mieszkańców.

✗ ZMYŚLONE + PĘTLA Kalifornia, Liverpool, „hrabstwo Boliwii" — fakty z sufitu, na końcu zacina się w pętli liczb.

Dawno, dawno temu…

– O czym ty mówisz? – To nie ma nic wspólnego z tym, że jest naprawdę zły. Nie mogę uwierzyć, że to zrobiłeś. Wyglądasz na zdenerwowanego. – Nie chcę tego robić. – Ale ja też.

✓ SKŁADNIA OPANOWANA Naturalny, poprawny dialog — model nauczył się stylu napisów filmowych. Zero błędów językowych

Wczoraj wieczorem poszedłem do…

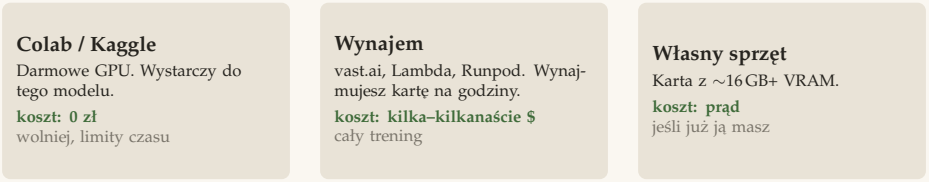
…domu, aby nie widzieć się z kim się spotkam. To było w porządku. – Zadzwonię później. – Dobra. Co ty robisz? Nie chcę tego robić. – Co się z tobą dzieje? – Nic.

✓ PŁYNNA POLSZCZYZNA Płynne zdania, naturalny rytm rozmowy. Treść błaha, ale język bez zarzutu.

Wniosek: składnia opanowana, wiedza zmyślona. Mały model uczy się JAK pisać po polsku, zanim nauczy się CO jest prawdą — a przy 124M to drugie ledwie ruszyło.

Model, który **pisze świetną polszczyzną i zmyśla fakty**. Zobaczysz na żywo, czemu tak jest — i to jest najważniejsza lekcja całego kursu.

1.5 SEKCJA 3 — Sprzęt i koszt (7:00–8:30)



Cały trening: godziny, nie tygodnie. Koszt rzędu jednej pizzy.

Nie potrzebujesz klastra. Trenowałem to na **jednym GPU**. Masz trzy drogi: - **Colab / Kaggle** — darmowe GPU, wystarczy do tego modelu (wolniej). - **Wynajem** (vast.ai, Lambda, Runpod) — kilka–kilkanaście dolarów za cały trening. - **Własny sprzęt** — jeśli masz kartę z ~16 GB+.

Cały trening tego modelu to godziny, nie tygodnie, i koszt rzędu jednej pizzy. W odcinku 5 pokażę dokładną konfigurację i pułapki sprzętowe (bo były).

1.6 ZADANIE (8:30–9:30)

(*wizual: git clone repo, struktura* `lab/`)

Twoje zadanie na ten odcinek jest proste: 1. Sklonuj repo kursu. 2. Postaw środowisko (`lab/README.md`). 3. Zdobądź dostęp do jednego GPU (wybierz drogę z tabelki).

Nic nie trenujemy jeszcze. Tylko przygotowanie. W następnym odcinku zaczynamy od danych — bo **model jest tylko tak dobry, jak tekst, którym go karmisz**.

1.7 CLIFF (9:30–10:00)

W odcinku 2: skąd wziąć gigabajty polskiego tekstu, jak go wyczyścić — i jak **nie** oszukać samego siebie, wpuszczając do treningu dane, na których potem testujesz. Pokażę ci prawdziwy bug, przez który **dwa z sześciu moich korpusów dały zero tokenów**, a tego nawet nie zauważyłem od razu.

Do zobaczenia.

1.7.1 Notatki produkcyjne

- Slajdy do dorobienia: `trzy-poziomy.png`, `koszt-sprzet.png`. Reszta gotowa w `slides/`.
- Długość napisów-haseł: trzymać ≤7 słów (pogrubienia w tekście).
- B-roll: nagrać `ls` repo + jedno przewinięcie krzywej uczenia jako teaser.

2 Odcinek 3 — Własny tokenizer (BPE od zera)

Czas: ~12 min **Cel:** Widz rozumie, czym jest tokenizer, dlaczego GPT-2 tnie polski na bajty, jak działa byte-level BPE — i sam trenuje własny tokenizer na polskim korpusie.

2.1 HAK (0:00–0:20)

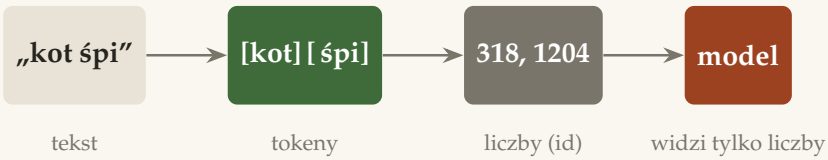
(*wizual: surowy* `polish_bpe_32k.json` *przewijany — pełno* `stÄĻp`, `ĠktÃ³ry`, `jÄħc`)

Otworzysz swój tokenizer i pomyślisz: zepsułem go. Wygląda jak śmieci.

Nie zepsułeś. To jest dokładnie tak, jak ma wyglądać — i do końca tego odcinka będziesz wiedział, dlaczego. A po drodze zbudujesz własny, który czyta polski **2.8 raza** wydajniej niż GPT-2.

2.2 INTRO — co to w ogóle tokenizer (0:20–2:00)

Model nie widzi liter. Widzi liczby. Tokenizer to słownik, który zamienia tekst na ciąg liczb (tokenów) i z powrotem.



Tokenizer to słownik: tekst ↔ liczby. Wybierasz go RAZ, przed treningiem — i już nie zmienisz.

I tu kluczowa rzecz, którą większość ludzi pomija: **tokenizer wybierasz ZANIM zaczniesz trenować, i już go nie zmienisz.** Jeśli źle podzieli polski, model do końca życia płaci ten podatek — wolniej się uczy, mieści mniej tekstu w kontekście.

Tokenizer to fundament. Dlatego robimy go sami.

2.3 SEKCJA 1 — Dlaczego GPT-2 nie nadaje się do polskiego (2:00–4:30)

Weźmy jedno zdanie: *„Wczoraj kupiłem książkę o województwie śląskim."*

- **GPT-2: 35 tokenów.** Połowa to — bo GPT-2 był trenowany na angielskim i nie zna liter ł, ś, ż, ą. Każdą rozbija na pojedyncze bajty.
- **Nasz: 10 tokenów.** Całe polskie słowa = jeden token: `województwie`, `książkę`.

(*wizual: terminal — odpalamy porównanie na żywo*)

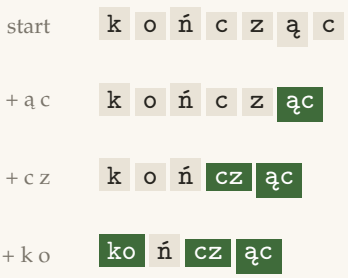
To nie jest kosmetyka. Zmierzmy gęstość na prawdziwym akapicie:

Nasz tokenizer: 5.22 znaku na token. GPT-2: 1.85.

Czyli na ten sam polski tekst GPT-2 zużywa **2.8× więcej tokenów**. A token to jednostka pracy modelu — więcej tokenów znaczy: dłuższy trening, droższy trening, mniej tekstu mieszczącego się w kontekście. **Własny tokenizer to najtańszy sposób, żeby model lepiej radził sobie z polskim.**

2.4 SEKCJA 2 — Jak to działa: BPE i „krzaki" (4:30–8:00)

Skąd model wie, że `województwie` to jeden token, a `najmocniejszy` to dwa? Z algorytmu **BPE — Byte Pair Encoding**. Idea jest banalnie prosta:



Reguła: najczęstsza sąsiednia para → nowy token → powtórz × tysiące razy.

Najczęstsze cząstki języka same dostają własny token — tokenizer uczy się ze statystyki.

> Zacznij od pojedynczych znaków. Znajdź najczęstszą sąsiadującą parę. Sklej ją w nowy symbol. Powtórz X tysięcy razy.

Najczęstsze cząstki języka (`ie`, `nie`, `ło`, `województw`) same wypływają na wierzch i dostają własny token. **Tokenizer uczy się struktury języka wyłącznie ze statystyki.**

2.4.1 Kto to wymyślił

- **1994 — Philip Gage:** BPE jako algorytm *kompresji danych*. Zero wspólnego z AI.
- **2016 — Sennrich, Haddow, Birch:** przenoszą BPE do NLP, do dzielenia słów na podsłowa.
- **2019 — OpenAI (GPT-2):** puszczają BPE nie na znakach, lecz na **bajtach** — i dorzucają mapowanie `bytes_to_unicode`. To stąd te „śmieci".

2.4.2 Skąd „krzaki"

(*wizual: tabelka zapis → znaczenie*)

Byte-level znaczy, że tokenizer pracuje na bajtach UTF-8, nie na literach. Żeby plik był czytelny i bezpieczny (bez spacji i znaków kontrolnych), każdy bajt dostaje widzialny zastępnik. Stąd:

w pliku	naprawdę	bo
`Ġ`	spacja	bajt 0x20 → widzialny znak
`stÄĻp`	`stęp`	ę to 2 bajty UTF-8 → 2 znaki
`jÄħc`	`jąc`	końcówka imiesłowów
`ĠktÃ³ry`	`który`	ó = `Ã³`

*(wizual: `tok.decode([512)*)` → `'stęp']`

To jest w 100% odwracalne. `decode()` zamienia krzaki z powrotem na czysty polski. A geniusz tego rozwiązania: skoro działamy na bajtach, tokenizer obsłuży **dowolny** tekst — każdy język, emoji, cokolwiek — i **nigdy** nie trafi na „nieznany znak".

2.5 SEKCJA 3 — Budujemy własny (8:00–10:30)

(*wizual: edytor — `lab/train_tokenizer.py`*)

Cała robota to kilkanaście linii dzięki bibliotece `tokenizers` od HuggingFace:

```
from tokenizers import Tokenizer, models, trainers, pre_tokenizers, decoders

tok = Tokenizer(models.BPE())
tok.pre_tokenizer = pre_tokenizers.ByteLevel(add_prefix_space=False)  # byte-level, jak GPT-
2
tok.decoder = decoders.ByteLevel()

trainer = trainers.BpeTrainer(
    vocab_size=32768,                                    # nasz słownik
    special_tokens=["<|endoftext|>"],                    # token końca dokumentu
    initial_alphabet=pre_tokenizers.ByteLevel.alphabet()# wszystkie 256 bajtów na start
)
tok.train(["polish_corpus.txt"], trainer)                # uczy się na TWOIM tekście
tok.save("polish_bpe_32k.json")
```

(*wizual: odpalenie treningu — przewija się przez korpus, zapisuje json*)

Dwie decyzje, które realnie mają znaczenie: - **`vocab_size`** — większy = krótsze sekwencje, ale więcej parametrów w warstwie embedding. 32768 to dobry kompromis dla jednego języka. - **Korpus** — tokenizer ssie statystykę z tego, co mu dasz. Trenuj go na **polskim**, inaczej dostaniesz GPT-2 od nowa.

2.6 DOWÓD (10:30–11:00)

(*wizual: porównanie po treningu*)

Sprawdzamy nasz świeży tokenizer na całych polskich słowach:

```
Warszawa        nasz: 1 token   GPT-2: 3
województwie    nasz: 1          GPT-2: 8
Rzeczpospolita  nasz: 1          GPT-2: 7
pięćdziesiąt    nasz: 1          GPT-2: 11
```

Jedno słowo, jeden token. **To jest cała różnica — i zrobiłeś to sam.**

2.7 ZADANIE (11:00–11:40)

(*wizual:* `lab/`)

1. Weź kawałek polskiego tekstu (Wikipedia, książka z Wolnych Lektur — cokolwiek).
2. Odpal `lab/train_tokenizer.py` na swoim korpusie.
3. Zmierz `znaki / token` dla swojego tokenizera i dla GPT-2 (`tiktoken`). **Cel: pobić 1.85 GPT-2.** Pochwal się wynikiem.

Eksperyment: zmień `vocab_size` na 8000 i 50000 — zobacz, co robi z gęstością.

2.8 CLIFF (11:40–12:00)

Mamy czym karmić model: tekst zamieniony na liczby. W następnym odcinku zaglądamy **do środka samego modelu** — embedding, mechanizm uwagi, bloki transformera. Rozłożymy GPT-2 na części i zobaczysz, że tam naprawdę nie ma magii. Tylko mnożenie macierzy.

Do zobaczenia.

2.8.1 Notatki produkcyjne

- Slajdy gotowe: `slayer-tokenizer.png`. Do dorobienia: diagram „tekst→token", animacja BPE, oś czasu (Gage/Sennrich/GPT-2).
- DOWÓD i porównania odpalić na żywo z `lab/` — liczby są prawdziwe, nie czytać z kartki.
- Tabelka „krzaki" — duże napisy, to najbardziej zapadająca w pamięć część odcinka.

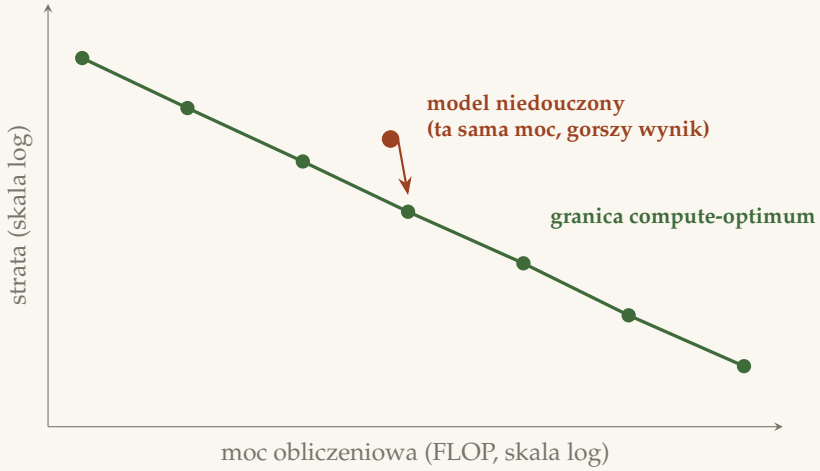
3 Odcinek 9 — Prawa skalowania (łagodne wejście w naukę o LLM)

Czas: ~12 min **Cel:** Widz rozumie, że jakość modelu jest *przewidywalna* (prawo potęgowe), zna regułę Chinchilli i sufit wiedzy — i potrafi umiejscowić swój mały model na mapie prawdziwych badań. To most z „zabawki" do frontiera.

Materiały: `handout-09-scaling.pdf`, figura `figures/scaling.pdf`, `course/LITERATURA.md`.

3.1 HAK (0:00–0:20)

Prawo skalowania: więcej mocy ⇒ niższa strata (przewidywalnie)



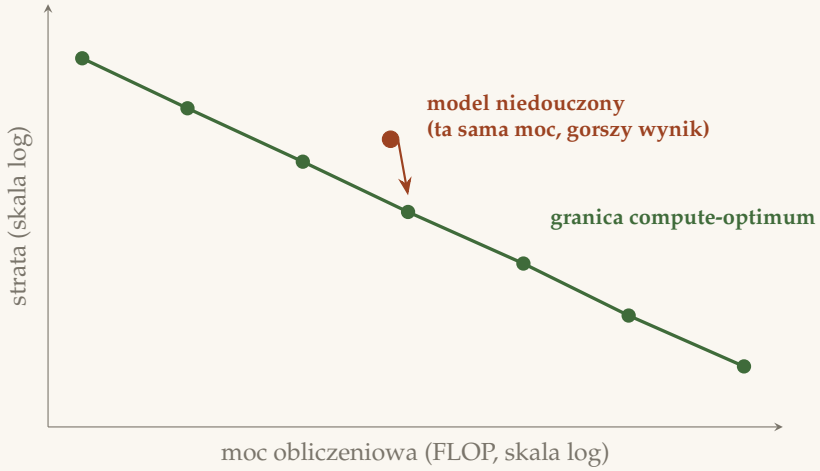
Kaplan i in. 2020 • Hoffmann i in. 2022 (Chinchilla): optimum ≈ 20 tokenów/parametr.

Zbudowałeś jeden mały model. A co, gdybyśmy zrobili go tysiąc razy większym?

Najlepsze jest to, że nie musimy zgadywać. Jakość większego modelu da się **przewidzieć z wyprzedzeniem** — i w tym odcinku zobaczysz, jak. To jest moment, w którym z „bawię się modelem" przechodzisz do „rozumiem, jak działa ten przemysł".

3.2 SEKCJA 1 — Strata jest przewidywalna (0:20–3:30)

Prawo skalowania: więcej mocy ⇒ niższa strata (przewidywalnie)



Kaplan i in. 2020 • Hoffmann i in. 2022 (Chinchilla): optimum ≈ 20 tokenów/parametr.

Najważniejsze odkrycie ostatniej dekady w ML brzmi nudno: **strata spada według prawa potęgowego.** Na wykresie log–log to zwykła **prosta linia**.

$$L(C) \approx a \cdot C^{(-b)}$$

Co to praktycznie znaczy? **Zanim wydasz milion dolarów na trening, możesz policzyć, jak dobry wyjdzie model.** Dokładnie tak planują modele OpenAI, DeepMind, Anthropic — nie metodą prób i błędów, tylko ekstrapolując prostą.

(*wizual: handout, wzór + wykres*)

□ **W literaturze:** Kaplan i in. 2020 — pierwsze prawa skalowania.

3.3 SEKCJA 2 — Reguła Chinchilli (3:30–6:30)

tokeny ≈ 20× parametry

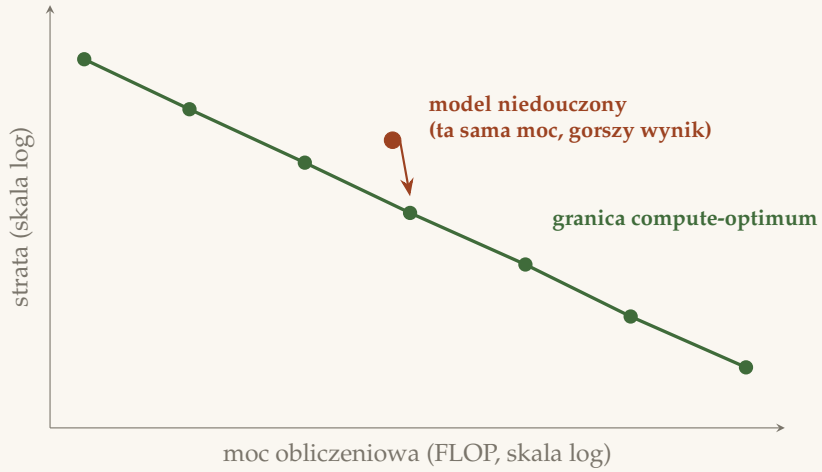
compute-optimum (Hoffmann i in. 2022)

Twój model: 124 M param ⇒ optimum ≈ **2,5 mld tokenów**.
Wytrenowane ≈ 0,84 mld ⇒ **∼3× za krótko**.

Skoro większy = lepszy, to po prostu róbmy gigantyczne modele? Nie. Hoffmann i in. (2022, „Chinchilla") pokazali, że dla danej mocy istnieje **optymalna para**: rozmiar modelu **i** liczba tokenów. Reguła kciuka:

tokeny ≈ 20 × parametry

Prawo skalowania: więcej mocy ⇒ niższa strata (przewidywalnie)



Kaplan i in. 2020 • Hoffmann i in. 2022 (Chinchilla): optimum ≈ 20 tokenów/parametr.

I teraz coś osobistego. **Twój model na tej mapie:** - 124M parametrów → optimum ≈ **2.5 mld tokenów**, - wytrenowałeś ≈ **0.84 mld** → jesteś **~3× za krótko**.

To nie wada — to po prostu za mało treningu. Czerwona kropka na wykresie to ty: ta sama moc, gorszy wynik, bo niedouczony.

□ **W literaturze:** Hoffmann i in. 2022. Powtórki danych? Muennighoff i in. 2023 — do ~4 epok jest OK.

3.4 SEKCJA 3 — Ale wiedza ma twardy sufit (6:30–8:30)

Skalowanie leczy płynność. Wiedzę — tylko częściowo. Allen-Zhu i Li (2024) zmierzyli pojemność: **~2 bity wiedzy na parametr.**

Wiedza ma fizyczny sufit: ≈ 2 bity / parametr (Allen-Zhu & Li 2024)

124 M parametrów × 2 bity ≈ **31 MB faktów** — tyle zmieści twój model.

Dlatego mały model zmyśla: fizycznie nie ma gdzie pomieścić wiedzy. Większy model to jedyne wyjście.

Twoje 124M to teoretycznie **~31 MB faktów** — ułamek Wikipedii. **Dlatego model zmyśla: fizycznie nie ma gdzie zmieścić wiedzy.** Żeby model „wiedział", musi być większy. Kropka. To nie kwestia lepszego promptu czy dłuższego treningu.

□ **W literaturze:** Allen-Zhu & Li 2024 (Part 3.3 — pojemność; Part 3.1 — wydobywanie wiedzy).

3.5 SEKCJA 4 — Jak teraz czytać prawdziwe papery (8:30–10:30)

(*wizual: otwarty arXiv, scroll po abstrakcie i rysunkach*)

Skoro znasz już język (strata, parametry, tokeny, skalowanie) — możesz czytać oryginały. Nie od deski do deski. Kolejność:

1. **Abstract** — o co chodzi w 5 zdaniach.
2. **Rysunki + podpisy** — 80% wniosków siedzi na wykresach.
3. **Lista wkładów** (koniec wstępu) i **Wnioski**.
4. Metoda — tylko jeśli naprawdę musisz.

Nie rozumiesz wzoru? Zrozum wykres i zdanie pod nim. Tak czytają badacze. Pełna, anotowana lista startowa: `course/LITERATURA.md`.

3.6 ZADANIE (10:30–11:30)

Zobacz prawo skalowania **na własne oczy**, na malutkim budżecie:

1. Wytrenuj ten sam model w 2–3 rozmiarach (np. n_embd 128 / 384 / 768), każdy na tej samej liczbie tokenów.
2. Zapisz końcowy val loss każdego.
3. Narysuj loss vs liczba parametrów na osi log–log.

Zobaczysz prostą. To samo prawo, co rządzi GPT-4 — na twoim laptopie.

3.7 CLIFF (11:30–12:00)

W finale: wracamy do naszego modelu i mówimy **brutalną prawdę** — co 124M naprawdę umie, czego nigdy nie osiągnie, i co byś musiał zrobić, żeby zbudować coś, co naprawdę zna polski. Spojlerek: to już nie jest weekendowy projekt.

Do zobaczenia.

3.7.1 Notatki produkcyjne

- Materiał wiodący: `handout-09-scaling.pdf` (pokazywać na ekranie obok narracji).
- Figury TikZ: `figures/scaling.pdf`. Do dorobienia: kafelek „20 tok/param", rachunek „31 MB".
- Sekcja 4 (jak czytać paper) to serce „łagodnego wejścia" — nie ucinać.

4 Literatura — od zabawkowego modelu do prawdziwej nauki o LLM

Ten kurs nie jest oderwany od badań. Każde zjawisko, które zobaczysz na własnym modelu 124M, jest **udokumentowanym prawem** w literaturze. Ta lista to twoja łagodna rampa do czytania prawdziwych prac.

4.1 Jak czytać paper (dla kogoś, kto nigdy nie czytał)

Nie czyta się od deski do deski. Kolejność:

1. **Abstract** — o co chodzi, w 5 zdaniach.
2. **Rysunki i ich podpisy** — 80% wniosków siedzi na wykresach. Czytaj je przed tekstem.
3. **Ostatni akapit wstępu** — zwykle lista „nasze wkłady" (contributions).
4. **Wnioski** (Conclusions) — co z tego wynika.
5. Dopiero potem metoda i dowody — jeśli ci naprawdę zależy.

Nie rozumiesz wzoru? Trudno. Zrozum **wykres i zdanie pod nim**. Tak czytają badacze.

4.2 Filar 1 — Skalowanie: większy/dłużej = lepiej (przewidywalnie)

- **Kaplan i in. 2020 —** ***Scaling Laws for Neural Language Models*** (arXiv:2001.08361) Pierwsze prawa potęgowe: strata spada przewidywalnie z parametrami, danymi i compute. Na log-log to prosta linia. To stąd cały przemysł wie, że „większe będzie lepsze".
- **Hoffmann i in. 2022 —** ***Training Compute-Optimal LLMs*** **(Chinchilla)** (arXiv:2203.15556) Optimum to **~20 tokenów na parametr**. Większość modeli była niedouczona. → Twoje 124M chce ~2.5B tokenów; zrobiłeś 0.84B, więc jesteś ~3× za krótko.

4.3 Filar 2 — Dane: kiedy brakuje, powtarzaj

- **Muennighoff i in. 2023 —** ***Scaling Data-Constrained Language Models*** (NeurIPS) (arXiv:2305.16264) **Do ~4 epok** powtarzania danych daje prawie ten sam efekt co świeże dane. → Wolno ci powtórzyć polski korpus, żeby dobić do Chinchilli.

4.4 Filar 3 — Wiedza ma fizyczny sufit

- **Allen-Zhu & Li 2024 —** ***Physics of LM, Part 3.3: Knowledge Capacity*** (ICLR'25) (arXiv:2404.05405) Model mieści **~2 bity wiedzy na parametr**. Twoje 124M → teoretycznie ~31 MB faktów, ułamek Wikipedii. → Wiedzy nie wyciśniesz z małego modelu. Kropka.
- **Allen-Zhu & Li —** ***Physics of LM, Part 3.1: Knowledge Storage and Extraction*** (arXiv:2309.14316) Model może *mieć* fakt, a nie umieć go *wypowiedzieć*, jeśli nie widział go w wielu formach. → To twój „paradoks Warszawy": zna w likelihood, nie generuje.

4.5 Filar 4 — Składnia szybko, wiedza późno (główna teza kursu)

- **Chang & Bergen 2024 —** ***Language Model Behavior: A Survey*** (MIT Press / Computational Linguistics) (link) Składnia uczona w ~pierwszych 20% treningu i nasycona; wiedza rośnie powoli przez cały czas. → 1:1 twoja krzywa gramatyka-vs-wiedza.
- **Mahowald i in. 2024 —** ***Dissociating language and thought in LLMs*** (arXiv:2301.06627) Kompetencja **formalna** (gramatyka, wcześnie) vs **funkcjonalna** (wiedza/rozumowanie, później). → Rama narracyjna całego kursu.
- **„How do language models learn facts?" 2025** (arXiv:2503.21676) Fakty pojawiają się późno i nierówno — stąd halucynacje małego/wczesnego modelu.
- **Warstadt i in. 2020 —** ***BLiMP*** (arXiv:1912.00582) Pary minimalne (zdanie poprawne vs z błędem) — metoda, którą mierzymy gramatykę w `lab/`.

Zasada kursu: **żadnej tezy bez dowodu.** Dowód = albo liczba z twojego modelu, albo paper z tej listy. Najlepiej oba.