

04834580 Software Engineering (Honor Track) 2025-26

# LLMs for Code

Sergey Mechtaev

`mechtaev@pku.edu.cn`

School of Computer Science, Peking University



## Definition (LLM)

A large language model (LLM) is a deep learning neural network, typically based on the transformer architecture [1], trained on vast text corpora to learn statistical patterns in language and generate, comprehend, and manipulate human-like text in response to input prompts.

*Source code is just another kind of text* — so the same models that predict the next English word can predict the next line of a program.

## **Inline completion**

Copilot, Cursor:  
finish the next line or function

## **Chat assistants**

explain, refactor,  
write tests, answer questions

## **Autonomous agents**

read a repo, edit files,  
run tests, iterate

## **Review & repair**

find bugs, localize faults,  
propose fixes

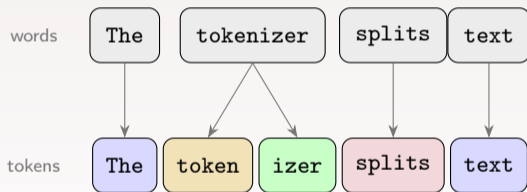
Everything an LLM does reduces to one deceptively simple task:

*Given a sequence of tokens, predict the next one.*

```
def is_even(n): return n % 2 == 0
```

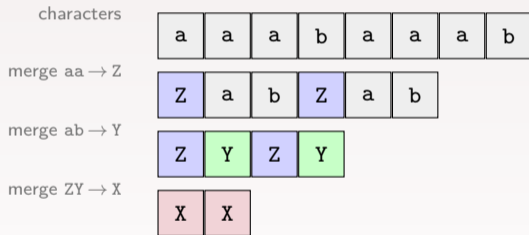
Writing a function, fixing a bug, holding a conversation — all of it **emerges** from doing this one thing, over and over, with a model trained on enough data.

A model never sees letters or whole words — it sees **tokens**: frequent chunks of text, each mapped to a number.



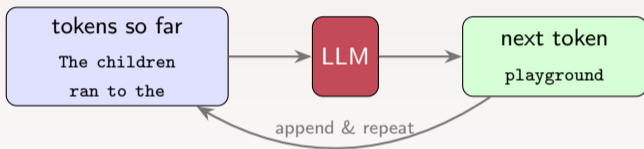
Common words are a single token; a rarer word like `tokenizer` splits into subword pieces — so the model can spell out *anything*.

Where do tokens come from? **BPE** [2] starts from characters and repeatedly *merges the most frequent adjacent pair*.



Frequent sequences collapse into single tokens. A vocabulary of  $\sim 100k$  tokens balances short sequences against rare words.

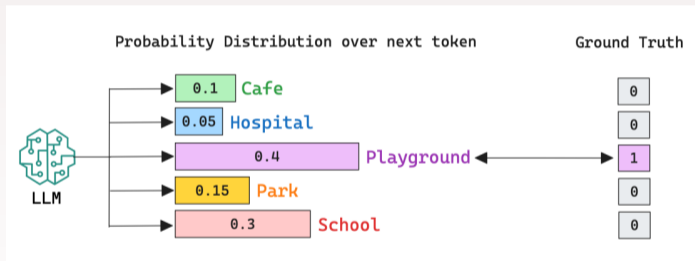
The model maps the tokens so far to a guess for the next one, then **feeds its own output back in** and repeats — this is what *autoregressive* means.



Formally, the probability of a whole sequence factors into one next-token decision after another:

$$P(x_1, \dots, x_T) = \prod_{t=1}^T P(x_t \mid x_1, \dots, x_{t-1})$$

At each step the model does not pick a single word — it assigns a probability to *every* possible next token.



We then choose one: take the most likely token (**greedy**), or **sample** from the distribution for more variety.

Temperature  $T$  rescales the scores before they become probabilities:

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$



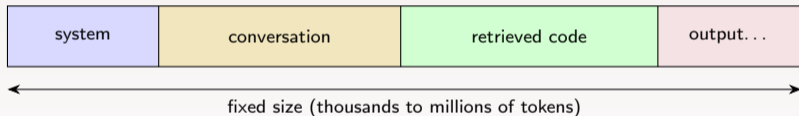
low  $T$ : *peaked*  
safe, repetitive



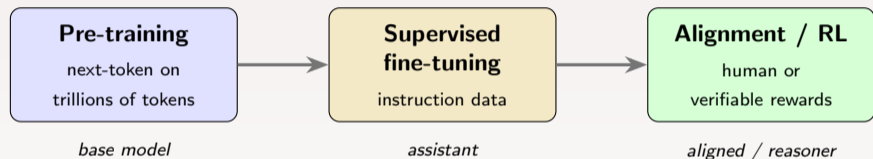
high  $T$ : *flat*  
diverse, creative

For code: low  $T$  for a single, careful answer; high  $T$  when sampling *many* candidates (we will see why with Pass@ $k$ ).

The model reads a fixed-size budget of tokens at once — its **context window**. Everything it knows for this query lives inside it.

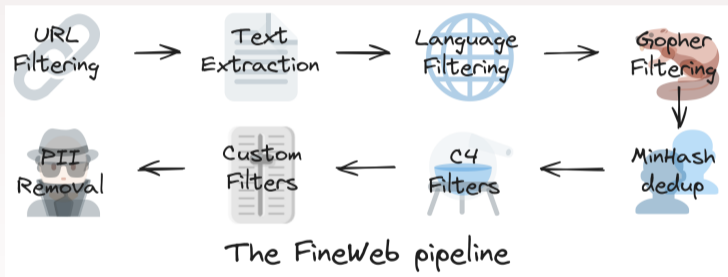


Anything outside the window is invisible; cost grows with length — which is why fitting the *right* code in matters.



**Pre-training** builds broad knowledge and language; the **post-training** stages teach the model to *follow instructions* and behave the way we want.

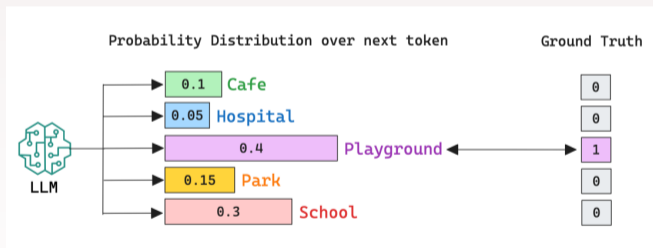
FineWeb [3] is a dataset of 15-trillion tokens (44TB disk space) of text collected from the internet.



The Stack 2 [4] adds 900B tokens (32TB) across 600+ programming languages. Raw data is heavily **filtered and deduplicated** — quality matters as much as quantity.

Slide a window over the corpus; at every position the model predicts the next token. The true token is the label — no humans needed (**self-supervised**).

The children ran to the | playground.



Training nudges the weights to raise the probability of the correct token and lower the rest.

Pre-training yields a **base model** — a superb *text completer*, but not yet a helpful assistant.

- ▶ Ask it a question and it may reply with *more questions* — it continues patterns, it does not “answer.”
- ▶ Yet broad skills have **emerged**: translation, arithmetic, coding — none explicitly taught.
- ▶ It can even learn a task *from the prompt alone*, with no weight updates: **in-context learning**.

LLMs extract a pattern from a few examples in the prompt and generalize — with *no* retraining [5].

Prompt

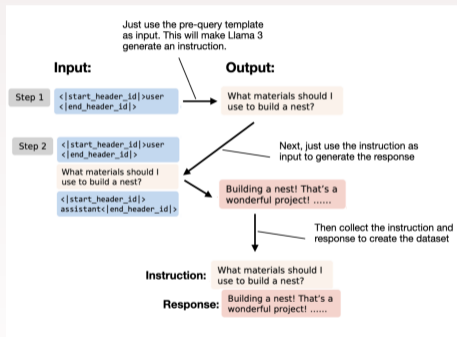
```
This is awesome! // Negative
This is bad! // Positive
Wow that movie was rad! // Positive
What a horrible show! //
```

Response

```
Negative
```

(The labels are deliberately flipped — the model follows the *demonstrated* mapping, not its prior.)

Fine-tune the base model on **(instruction, response)** pairs to create an assistant. Data may be human-written or synthetic, e.g. Magpie [6].



A **chat template** marks roles (system / user / assistant), teaching the model *when* it is its turn to speak.

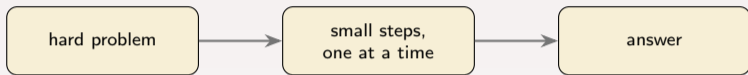
SFT shows the model good answers; **alignment** teaches it which answers people *prefer*.

- ▶ **RLHF** [7]: humans rank outputs; a reward model learns those rankings; the LLM is optimized to score well.
- ▶ **DPO** [8]: a simpler recipe that learns directly from preference pairs.

The goal: more *helpful, honest, and harmless* behaviour. This is what turns a capable text engine into a usable assistant.

Standard Prompting	Chain-of-Thought Prompting
<p><b>Model Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>	<p><b>Model Input</b></p> <p>Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?</p> <p>A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. <math>5 + 6 = 11</math>. The answer is 11.</p> <p>Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?</p>
<p><b>Model Output</b></p> <p>A: The answer is 27. ❌</p>	<p><b>Model Output</b></p> <p>A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had <math>23 - 20 = 3</math>. They bought 6 more apples, so they have <math>3 + 6 = 9</math>. The answer is 9. ✅</p>

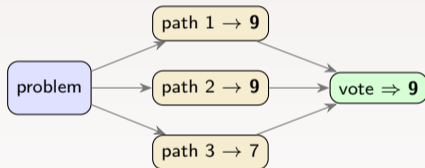
Letting the model show intermediate steps sharply improves multi-step reasoning. Zero-shot-CoT [9] triggers it by simply adding “Let’s think step by step.”



Each generated token is another pass through the network — so a longer chain literally buys the model **more computation** to reach an answer, and writes intermediate results into the context to reuse later.

*Quality now depends on test-time compute, not just training compute.*

Sample **many** reasoning paths, then take a **majority vote** [11]: routes that agree are more likely correct.

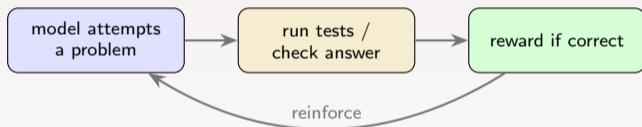


*Spend extra compute at inference time to buy accuracy.*

CoT and self-consistency are *prompting* tricks. Newer **reasoning models** (OpenAI o1, DeepSeek-R1) are *trained* to reason on their own.

- ▶ They generate a long internal “thinking” trace — exploring, checking, backtracking — before the final answer.
- ▶ The thinking happens *automatically*; the user need not ask.
- ▶ More thinking generally means higher accuracy on hard math, logic, and coding.

How do you *train* a model to reason well? For math and code, the answer can be **checked automatically**.

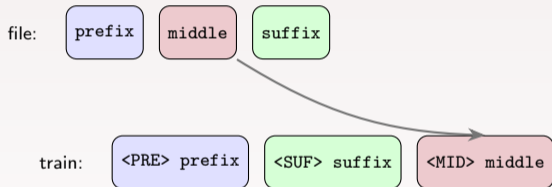


DeepSeek-R1 [12]: large-scale RL on verifiable problems *elicits* long chains of thought and self-checking — no step-by-step human supervision. (Contrast RLHF, whose reward is learned *human preference*.)

Same architecture — the *data* and *objectives* are tuned for code.

- ▶ Trained on huge code corpora (The Stack v2 [4]), *mixed* with natural-language text.
- ▶ Learns syntax, idioms, and APIs across hundreds of languages; the text keeps it able to read issues and *explain* its code.
- ▶ Special objectives match how code is *edited*, not just read top-to-bottom — next slide.

In an editor you usually complete code in the *middle* of a file — there is text both before *and* after the cursor. A left-to-right model cannot use the suffix.

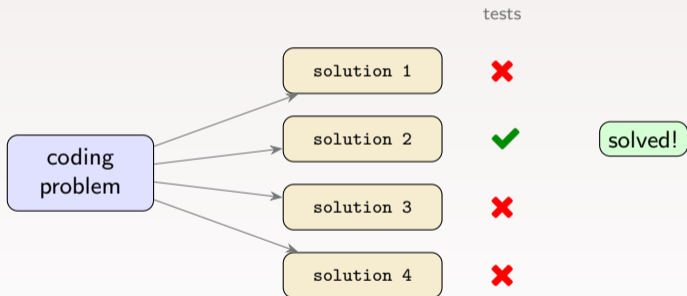


**Trick [13]:** during training, move the middle to the *end* with special tokens. The model still predicts left-to-right, but now conditions on both sides — powering IDE completion.

Real code does not live in a single file. To edit correctly, a model needs the *right* surrounding context placed into its window:

- ▶ **Retrieval (RAG)** — search the codebase for relevant definitions, usages, and tests.
- ▶ **Structure-aware prompts** — imports, type signatures, neighbouring functions.
- ▶ **Instruction tuning for code** — OSS-Instruct / Magicoder [14] turns *real* snippets into realistic (problem, solution) training pairs.

Let the model generate  $k$  independent solutions, then run each against the hidden tests. The problem counts as **solved** if at least one passes.

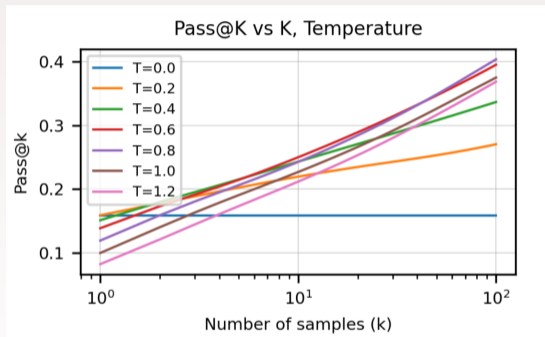


One of four works, so this problem is solved at  $k = 4$  — though it would *fail* at  $k = 1$ , since the first attempt was wrong.

**Pass@ $k$**  averages this over many problems: generate  $k$  samples per problem and count it solved if *any* sample passes the tests [15].

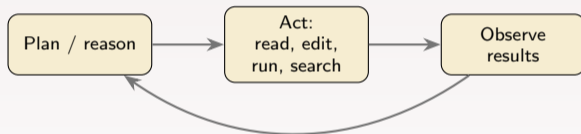
$$\text{Pass@}k = \mathbb{E} \left[ 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

where  $n$  samples are generated and  $c$  of them pass. Pass@1 reflects single-shot quality; higher  $k$  rewards a *diversity* of attempts.



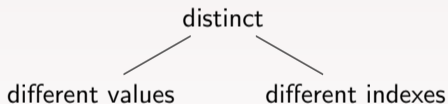
Low temperature wins for a single sample (Pass@1); higher temperature wins as  $k$  grows, because diverse samples cover more of the solution space.

**HumanEval** / **MBPP** [15]: short self-contained functions. **SWE-bench** [16]: real GitHub issues — patch a repo until its tests pass.



An **agent** loops the model with tools and *verifiable feedback* (the tests) — the engine behind today's SWE-bench leaders.

*A function that takes a list of integers as an input. It returns True if there are three **distinct** elements in the list that sum to zero, and False otherwise.*



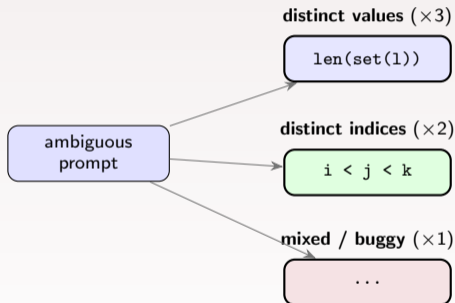
## Prompt

A function that takes a list of integers as an input. It returns True if there are three **distinct** elements in the list that sum to zero, and False otherwise.

## Response

```
if len(l) < 3 or len(set(l)) < 3:  
    return False  
for i in range(n-2):  
    for j in range(i+1, n-1):  
        for k in range(j+1, n):  
            if l[i]+l[j]+l[k]==0:  
                return True  
return False
```

Generation is *probabilistic*: ask the same ambiguous prompt many times and watch how the answers spread.



All answers *mean* the same thing  $\Rightarrow$  **confident**. Answers split across *different meanings*  $\Rightarrow$  **uncertain** — a red flag that the spec is ambiguous.

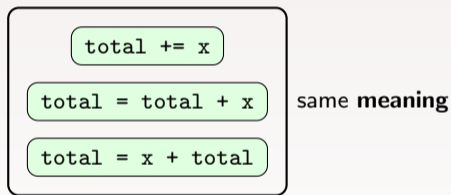
**Entropy** turns “how spread out” into one number — the *average surprise* of an outcome:

$$H(p) = - \sum_y p(y) \log p(y)$$



A sure thing carries no surprise ( $H = 0$ ); a coin flip carries one bit. Spread belief over many *conflicting* answers and entropy is high.

Taking the entropy of raw *token sequences* is misleading — the same behaviour can be written in countless ways:

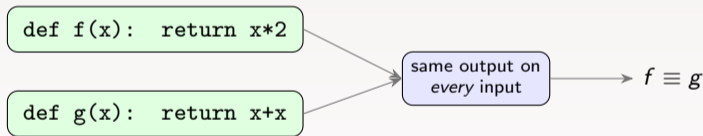


Different text, identical effect. So first **group answers by what they do** (e.g. run them and compare behaviour), then measure spread *across the groups*.

Grouping answers by meaning needs a notion of **program equivalence**.

## Definition (Observational equivalence)

Two programs are equivalent if they produce the *same output for every input*.



**But** deciding this exactly is *undecidable* (Rice's theorem). In practice we **approximate**  $\equiv$  by running both programs on a set of inputs and comparing behaviour — exactly how the meaning-classes are formed.

**Semantic entropy** [17]: cluster the sampled answers into meaning-classes, then take the entropy *over the classes*.

## Definition (Semantic Entropy, Kuhn et al. 2023)

Let  $m$  be an LLM,  $\equiv$  a semantic equivalence relation over responses, and  $m_{\equiv}$  the induced distribution over equivalence classes. The semantic entropy is

$$\text{SE}(m_{\equiv}(\cdot | x)) \triangleq - \sum_y m_{\equiv}(y | x) \log m_{\equiv}(y | x).$$

$m_{\equiv}(y | x)$  is the probability the model produces *meaning*  $y$ . **High SE** = the model is genuinely torn  $\Rightarrow$  flag the ambiguous spec, ask for clarification, or distrust the code.

Treat generated code like a junior contributor's pull request — always review, test, and verify.

- ▶ **Hallucination** — confidently invents nonexistent APIs.
- ▶ **Subtle bugs** — plausible code that is off-by-one or mishandles edge cases.
- ▶ **Security** — may reproduce vulnerable patterns; exposed to *prompt injection*.
- ▶ **Licensing & stale knowledge** — output may echo training code; frozen at the cutoff.

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.  
Attention is all you need.  
*Advances in neural information processing systems*, 30, 2017.
- [2] Rico Sennrich, Barry Haddow, and Alexandra Birch.  
Neural machine translation of rare words with subword units.  
*In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1715–1725, 2016.
- [3] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al.  
The fineweb datasets: Decanting the web for the finest text data at scale.  
*Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

- [4] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al.  
Starcoder 2 and the stack v2: The next generation.  
*arXiv preprint arXiv:2402.19173*, 2024.
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al.  
Language models are few-shot learners.  
*Advances in neural information processing systems*, 33:1877–1901, 2020.
- [6] Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin.  
Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing.  
*arXiv preprint arXiv:2406.08464*, 2024.

- [7] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al.  
Training language models to follow instructions with human feedback.  
*Advances in neural information processing systems*, 35:27730–27744, 2022.
- [8] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn.  
Direct preference optimization: Your language model is secretly a reward model.  
*Advances in Neural Information Processing Systems*, 36, 2023.
- [9] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa.  
Large language models are zero-shot reasoners.  
*Advances in neural information processing systems*, 35:22199–22213, 2022.

- [10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al.  
Chain-of-thought prompting elicits reasoning in large language models.  
*Advances in neural information processing systems*, 35:24824–24837, 2022.
- [11] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou.  
Self-consistency improves chain of thought reasoning in language models.  
*arXiv preprint arXiv:2203.11171*, 2022.
- [12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al.  
DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning.  
*arXiv preprint arXiv:2501.12948*, 2025.

- [13] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen.  
Efficient training of language models to fill in the middle.  
*arXiv preprint arXiv:2207.14255*, 2022.
- [14] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang.  
Magicoder: Empowering code generation with oss-instruct.  
*arXiv preprint arXiv:2312.02120*, 2023.
- [15] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al.  
Evaluating large language models trained on code.  
*arXiv preprint arXiv:2107.03374*, 2021.

- [16] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan.  
SWE-bench: Can language models resolve real-world GitHub issues?  
*arXiv preprint arXiv:2310.06770*, 2023.
- [17] Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar.  
Semantic uncertainty: Linguistic invariances for uncertainty estimation in natural language generation.  
*In International Conference on Learning Representations (ICLR)*, 2023.