AI 6001
Artificial Intelligence

Lecture 02

Intro to LLMs / AI Coding Tools

Quick Note

- We would all be capable of jumping immediately into a demo of AI tools
- We will first go over a background of LLMs and relevant terminology so you can better understand the whole process
- Understanding these terms and processes will lead to a better overall experience and outcome from these tools

What is an LLM?

- LLM stands for Large Language Model
- Machine learning model that is trained on massive amounts of text / code / data
- Learns statistical patterns in text, not a list of facts or numbers like a database
- Core job of an LLM is to predict the next token given previously seen tokens

What is Large? LLM

- Large usually refers to the very high parameter counts, compute required, and data used to train these modern models
- More data / compute / scale often leads to models with higher reasoning capability
- Larger is not always better, because we must factor in cost, speed, and latency

What is Language? LLM

- LLMs primarily get trained on text, and learn to predict text-based tokens
- Code is treated just like any other language with syntax, structure, patterns
- Modern systems are often multimodal, but the core models are still language based

What is a Model? LLM

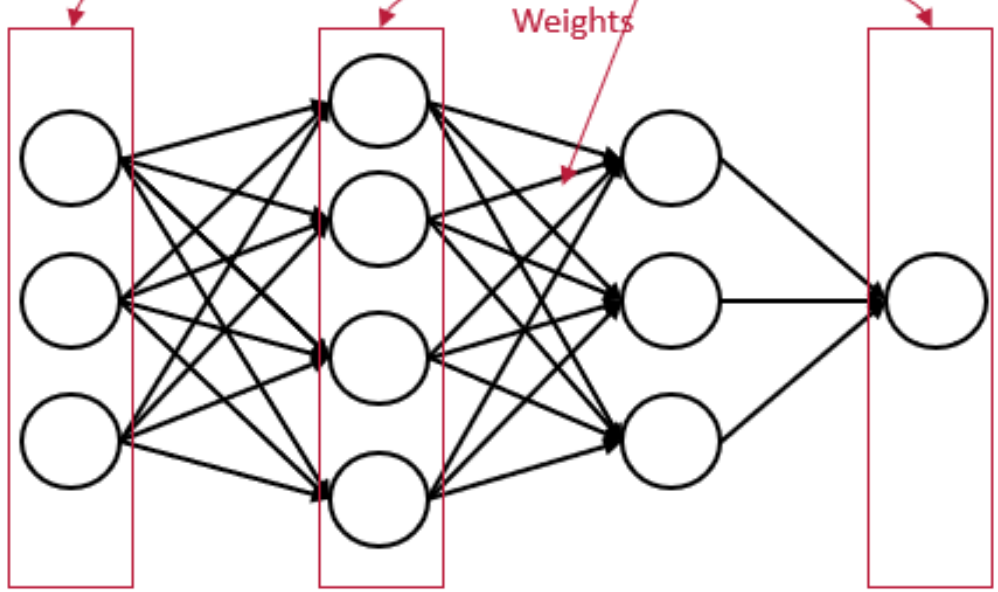
- A model is a learned function that maps input text to predicted output text
- Usually implemented as a neural network
- Models do not 'look up' information in the same was as a database or search engine
- Generates responses based on learned statistical patterns / relationships

Intro to Neural Networks

Model Size

- Model size refers to parameter count
  - Parameter = weights of underlying NN
- Larger models can capture more complex patterns and generally perform better
- Larger size means larger training and inference cost, higher memory and latency
- Model size is an important consideration

$$f(x) = g \circ f_K \circ \dots f_2 \circ f_1(x) \qquad z_i = w_i a_{i-1} + b_i \qquad a_i = \sigma(z_i)$$



https://pub.towardsai.net/backpropagation-2eeb25201095

Model Type	Parameters	Typical Hardware	Best For	Key Advantage
Small	<3B	Laptops, edge devices	Classification, sentiment, embeddings	Low latency and power use
Medium	7–13B	Consumer GPUs (8–24 GB)	Chatbots, summarization, RAG pipelines	Balanced accuracy and cost
Large	30–70B	Multi-GPU or cloud	Complex reasoning, multilingual QA	Higher contextual accuracy
Very Large	100B+	Enterprise clusters	Multimodal or research models	Advanced reasoning and creativity

https://labelyourdata.com/articles/llm-fine-tuning/llm-model-size

Why LLMs run on GPUs

- LLMs rely primarily on neural networks
- NN feed forward compute is dominated by linear algebra such as matrix multiplications
- They make the same calculations over very large arrays of values stored in the NN
- GPUs are designed to perform these type of operations in parallel, more cores than CPU

Why VRAM Matters

- LLM must be loaded into VRAM on GPU in order to run inference as efficiently as possible
- GPU must access entire model each inference
- VRAM is very fast, but also expensive
- If model fits in VRAM, the model will run quickly
- If it does not fit, offloaded to slower system RAM
- GPU VRAM directly constrains the size and speed of the model you can run efficiently

Local vs Cloud

- Models can be run in the cloud as a paid service, or locally on your own hardware (GPUs usually)
- Local models offer more privacy, control, and offline access, but limited to hardware you have
- Cloud models are usually much larger and capable of solving more complex problems
- Must balance upfront cost and lower capability of local models with more capable pay per use cloud

Model Quantization

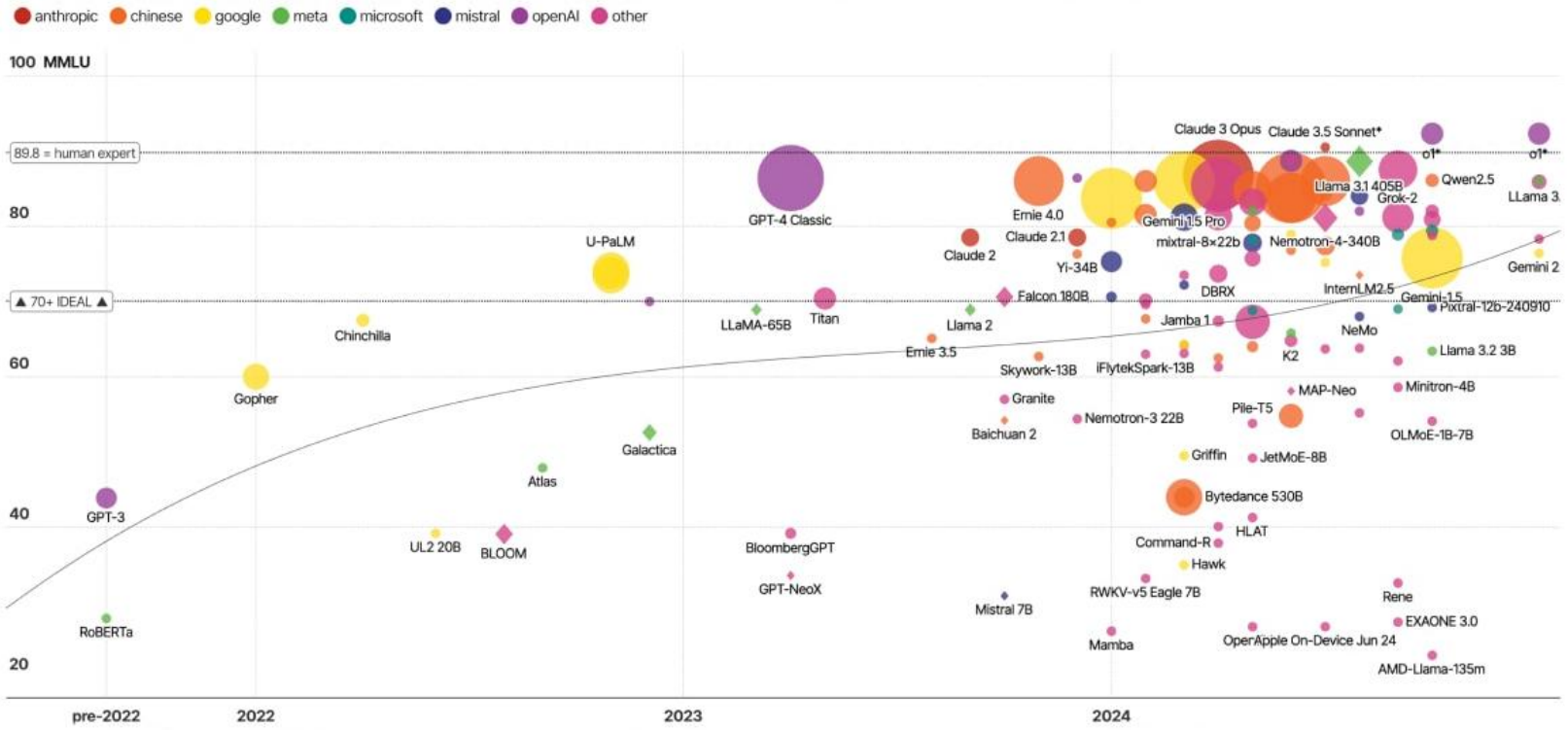
- Quantization is the process of storing a model with lower numerical precision
- High precision FP32 / FP16 can be quantized to fewer bits such as 8-bit, 6-bit, 4-bit
- Main goal is to reduce memory usage and sometimes speed up inference
- Lower precision models can lose quality

GPU	VRAM	CUDA	SU's	Tensor	TMU	RT Cores	SM's	ROP	Mem. Band.
5090	32 GB	21,760	21,760	680	680	170	170	176	1,792 GB/s
4090	24 GB	16,384	16,384	512	512	128	128	176	1,008 GB/s
3090 TI	24 GB	10,752	10,752	336	336	84	84	112	936 GB/s
3090	24 GB	10,496	10,496	328	328	82	82	112	936 GB/s
5090 Laptop	24 GB	10,496	10,496	328	328	84	84	128	936 GB/s
5080	16 GB	10,752	10,752	336	336	84	84	112	960 GB/s
4080 Super	16 GB	10,240	10,240	320	320	80	80	112	716 GB/s
4090 Laptop	16 GB	9,728	9,728	304	304	76	76	112	768 GB/s
4080	16 GB	9,728	9,728	304	304	76	76	112	716 GB/s
5070 TI	16 GB	8,960	8,960	280	280	70	70	128	896 GB/s
4070 TI Super	16 GB	8,448	8,448	264	264	66	66	96	672 GB/s
4070 TI	16 GB	7,680	7,680	240	240	60	60	80	504 GB/s
5080 Laptop	16 GB	7,680	7,680	240	240	60	60	60	811 GB/s
3080 TI Laptop	16 GB	7,424	7,424	232	232	58	58	96	512 GB/s
3080 Laptop	16 GB	6,144	6,144	192	192	48	48	96	448 GB/s
4060 TI	16 GB	4,352	4,352	136	136	34	34	48	288 GB/s
3080 TI	12 GB	10,240	10,240	320	320	80	80	112	912 GB/s
4080 Laptop	12 GB	7,424	7,424	232	232	58	58	80	504 GB/s
4070 Super	12 GB	7,168	7,168	224	224	56	56	80	504 GB/s
3080 - 12 GB	12 GB	8,960	8,960	280	280	68	68	96	912 GB/s
5070	12 GB	6,144	6,144	192	192	48	48	80	672 GB/s
5070 TI Laptop	12 GB	5,888	5,888	184	184	46	46	64	608 GB/s
4070	12 GB	5,888	5,888	184	184	46	46	64	504 GB/s
3080 - 10 GB	10 GB	8,704	8,704	272	272	68	68	96	760 GB/s
3070 TI	8 GB	6,144	6,144	192	192	48	48	96	608 GB/s
3070	8 GB	5,888	5,888	184	184	46	46	96	448 GB/s
3070 TI Laptop	8 GB	5,888	5,888	184	184	46	46	96	448 GB/s
3070 Laptop	8 GB	5,120	5,120	160	160	40	40	80	448 GB/s
5070 Laptop	8 GB	4,608	4,608	144	144	36	36	48	405 GB/s
4070 Laptop	8 GB	4,608	4,608	144	144	36	36	48	288 GB/s
4060 TI	8 GB	4,352	4,352	136	136	34	34	48	288 GB/s
4060 Laptop	8 GB	3,072	3,072	96	96	20	20	48	288 GB/s

Params	4-bit	6-bit	8-bit	FP16 BF16
1B	0.5 GB	0.75 GB	1 GB	2 GB
3B	1.5 GB	2.25 GB	3 GB	6 GB
7B	3.5 GB	5.25 GB	7 GB	14 GB
8B	4 GB	6 GB	8 GB	16 GB
13B	6.5 GB	9.75 GB	13 GB	26 GB
14B	7 GB	10.5 GB	14 GB	28 GB
32B	16 GB	24 GB	32 GB	64 GB
34B	17 GB	25.5 GB	34 GB	68 GB
70B	35 GB	52.5 GB	70 GB	140 GB
72B	36 GB	54 GB	72 GB	144 GB
100B	50 GB	75 GB	100 GB	200 GB
175B	87.5 GB	131.25 GB	175 GB	350 GB
400B	200 GB	300 GB	400 GB	800 GB

Approx required VRAM for model parameter size and quantization level

LLM Size vs. Performance



Comparing Models

		Features	Intelligence	Price	Speed	Latency	
Model	Context Window	Creator	Artificial Analysis Intelligence Index	Blended *USD/1M Tokens*	Median *Tokens/s*	Latency *First Answer Chunk (s)*	Further Analysis
Gemini 3.1 Pro Preview	1m	Google	57	$4.50	116	30.62	Model Providers
GPT-5.4 (xhigh)	1m	OpenAI	57	$5.63	84	163.63	Model Providers
GPT-5.3 Codex (xhigh)	400k	OpenAI	54	$4.81	71	89.92	Model Providers
Claude Opus 4.6 (max)	200k	Anthropic	53	$10.00	52	10.83	Model Providers
Claude Sonnet 4.6 (max)	200k	Anthropic	52	$6.00	63	81.66	Model Providers
GLM-5	200k	Z AI	50	$1.55	89	1.44	Model Providers
MiniMax-M2.7	205k	MiniMax	50	$0.53	46	2.24	Model Providers
MiMo-V2-Pro	1m	Xiaomi	49	$0.00	0	0.00	Model Providers
Grok 4.20 Beta 0309	2m	xAI	48	$3.00	205	12.33	Model Providers
GPT-5.4 mini (xhigh)	400k	OpenAI	48	$1.69	259	4.61	Model Providers

https://artificialanalysis.ai/leaderboards/models

Prompts

- Prompts are inputs given to the LLM
  - For coding problems, prompts should be specifications
- Quality of LLM output depends on:
  - How well you specify the task in general (be detailed)
  - Problem constraints and assumptions
  - Desired output structure and goal criteria
- Effective prompts look similar to task specifications you would give to another engineer for a project
- Ambiguous prompts can produce weak outputs

Tokens

- LLMs process text as tokens, not words
- Tokens are chunks of text
  - Whole or part of a word
  - Punctuation, white space
  - Code symbols or fragments
- Models do not see literal English words, they see a sequence of token IDs

Token Usage in LLMs

- Model looks at a previous sequence of tokens and predicts which token is likely to come next, which is the output
- This happens repeatedly, one by one
- The model output (sentence, code, etc) is generated as a long chain of next-token predictions that form a coherent message

Token Importance

- Cost of an LLM is typically based on how many input/output tokens are used
- Speed is usually discussed in tokens/sec
- Context size is measured in tokens, not in pages of text or English word count
- Large conversations consume context because they contain many tokens

Context Window / Size

- The context window of an LLM is the maximum number of tokens that can fit into the model's active working context for a given interaction
- This includes your prompt, chat history, code, input documents, and model reply
- If token count gets too large, older content can be discarded, truncated, compressed, summarized
- Larger context windows helps with more complex tasks, longer documents, or multi-step tasks

LLM System Cost

- Commercial systems often charge by token volume, some separating by input and output
- Output tokens typically more expensive than input
- Rereading files, resupplying context, letting an agent loop over the same content, etc, can cause the cost of using a system to grow unexpectedly
- Efficient usage relies on providing enough context to solve the problem but not too much that it wastes

Cost Comparison Example

Chat/Completion | Fine-tuning | Embedding

Search...

Provider	Model	Context	Input/1k Tokens	Output/1k Tokens	Per Call	Total
Chat/Completion						
Kimi	Kimi K2	128k	$0.001	$0.003	$0.0040	$0.40
OpenAI	GPT-5 nano	400k	$0.00005	$0.0004	$0.0005	$0.05
OpenAI	GPT-5 pro	400k	$0.015	$0.12	$0.1350	$13.50
OpenAI	GPT-5 mini	400k	$0.00025	$0.002	$0.0023	$0.23
OpenAI	GPT-5	400k	$0.00125	$0.01	$0.0112	$1.13
Anthropic	Claude Sonnet 4.5	200k	$0.003	$0.01	$0.0130	$1.30

https://yourgpt.ai/tools/openai-and-other-llm-api-pricing-calculator

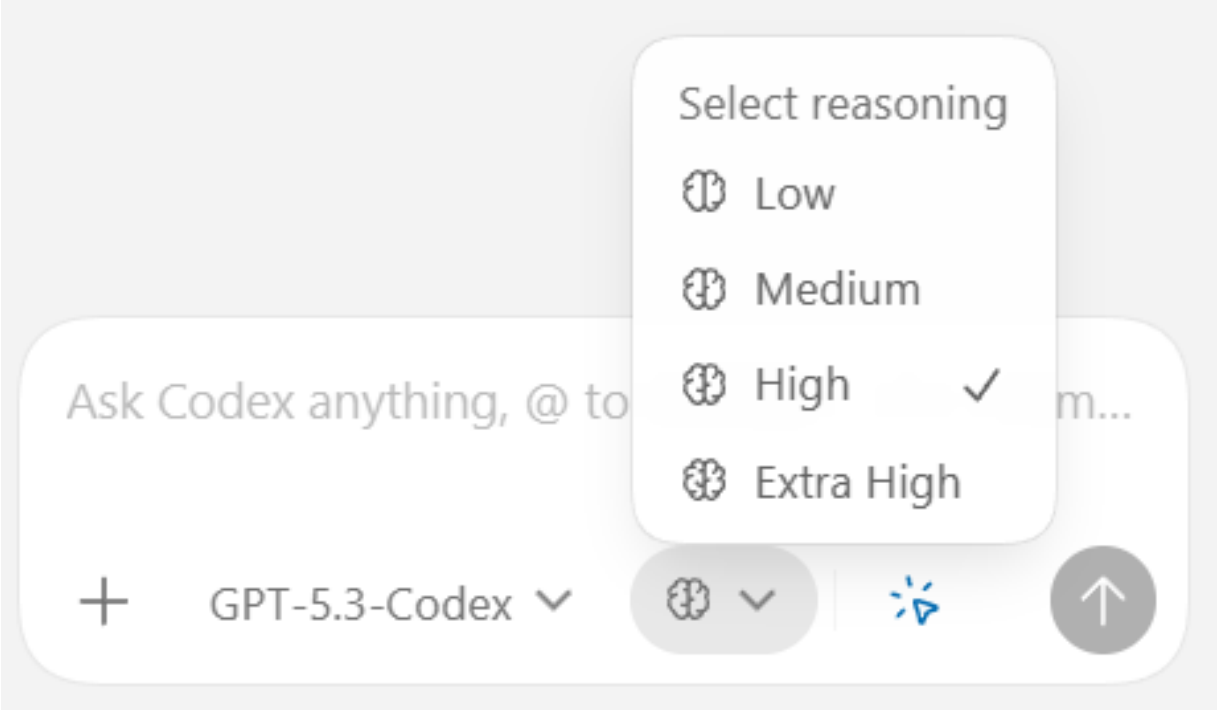
Throughput and Latency

- Latency is the delay between sending a request and the beginning of the response
- Throughput is the speed of text generation once started, in tokens/sec
- Both affect how system feels in practice
- Faster system interaction can matter more than slight benchmark differences

Model Reasoning Settings

- Modern AI systems may allow you to control how much reasoning effort the model uses
- Settings ex: fast / balanced / deep / low / high
- Higher thinking settings spend more computation and time on planning, checking, multistep reasoning
- Lower thinking settings produce faster results, but may reduce performance on complex tasks
- Balance these settings for your needs

Model Reasoning Settings



Reliability and Hallucination

- LLMs can produce outputs that are fluent, confident, coherent, but completely incorrect
- In software tasks, this can be:
  - Invented APIs / nonexistent functions
  - Incorrect assumptions about behavior
  - Unjustified confidence about correctness
- Self-verification is a dangerous assumption
- Human verification of final code is still a vital step to ensure system functionality and security

Dangers of Vibe Coding

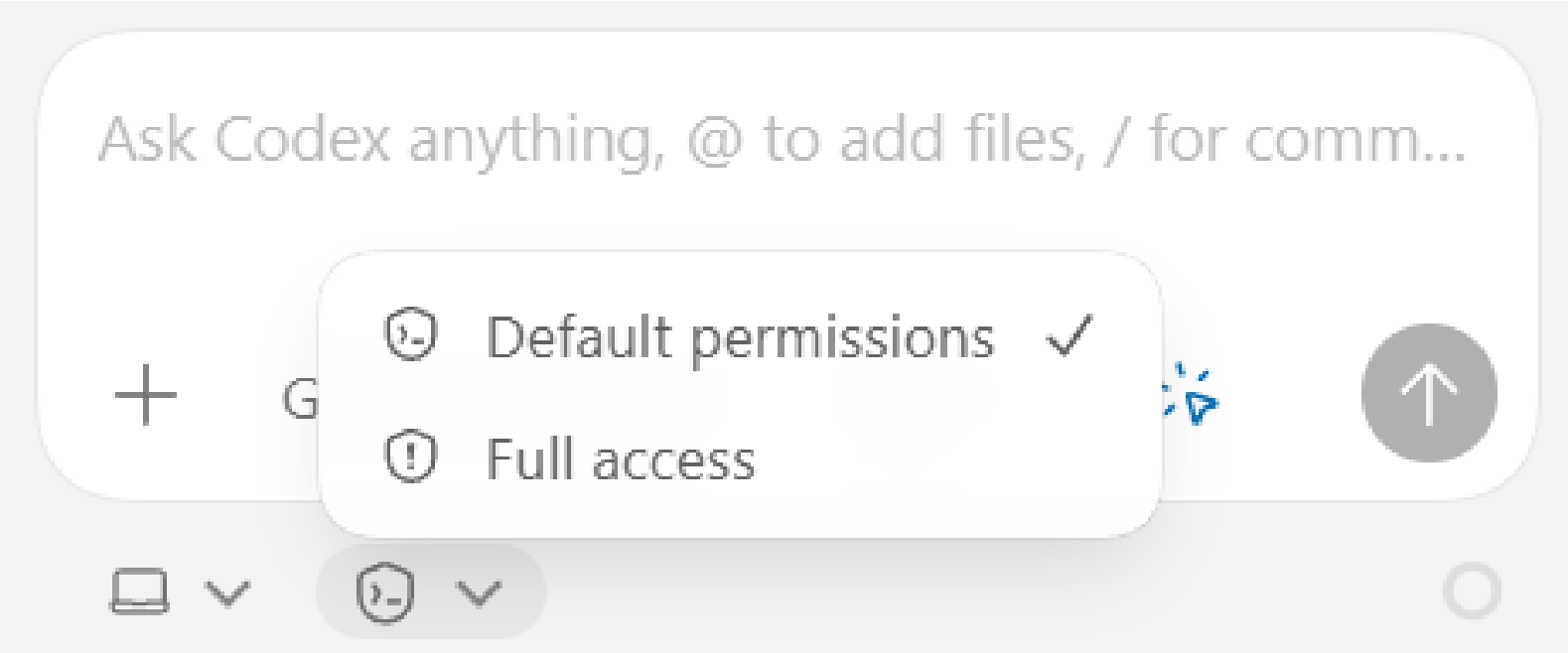
- Using AI tools without the necessary skill to verify their output and overall system stability is incredibly short sighted and dangerous
- AI code can look correct to newer programmers, but contain subtle bugs or incorrect logic that can lead to system instability or security risks
- This type of failure is not obvious, since the LLM desperately wants to please you and will always tell you its results are correct unless challenged

Human in the Loop

- AI coding tools should be treated as assistants to professionals, not authorities with the final say
- Humans must be in the loop to:
  - Define the task and desired outcomes
  - Judge whether output matches requirements
  - Design, run, and verify tests results

- A human should be responsible for verification of the output of the system, not the AI itself

Agent Permissions



Best Use Cases for AI Tools

- High-speed assistant for rapid prototyping
  - Going from nothing to something quickly
- Refactoring / cleaning up existing code
  - Eliminates the ‘grunt work’ of typing
- Summarizing / reviewing existing code
  - Suggesting optimizations / finding bugs
  - Doesn’t get tired of reading thousands of lines
- Brainstorming alternative implementations
- Obscure knowledge of poorly documented APIs

Worst Use Cases for AI Tools

- Trying to write large systems from scratch
- Large changes to existing code bases
  - Results in impossible to review PRs
  - No way to verify in a given time frame
- Trusting it with only its own test cases
- Giving agentic systems full control to modify code base however it wants

Best Practices

- Always ask a model to propose a plan before implementing it for a given task
- Have it identify root causes, affected files, and minimal safe changes before implementing
- Prefer minimal patches over large refactoring
- Ask it to explain why a change is needed
- State hard constraints explicitly and clearly
- Ask for alternatives and choose the best option

Final Thoughts

- These tools are best understood as probabilistic sequence prediction models embedded in a nice UI
- They promise everything is correct, but can make catastrophic mistakes disguised as good code
- With increasingly agentic functionality, their usefulness and accuracy depends on how well we manage context, constraints, cost, and verification
- Use these tools to speed up a given task, do not blindly trust them with mission critical functionality