LLMs

Advanced Computer Security
CS563 / ECE522

Nikita Borisov, Spring 2025

LLMs and Security

- LLMs used for a variety of tasks
  - Writing
  - Research
  - Programming
- What else?
  - Generate slides (such as this presentation!)
- **What are security implications?**

LLMs: A Brief History

From Rules to Reasoning

Rule-based NLP (Pre-1990s)

- Systems were explicitly programmed with linguistic rules
- Examples: ELIZA, SHRDLU
- Pros: Transparent, interpretable
- Cons: Brittle, didn’t scale well to real-world languages

Basic Machine Learning (1990s-2010s)

- Shift to statistical NLP
- Models learned from labeled data
- Examples: Naive Bayes, decision trees, logistic regression, CRFs
- Enabled tasks like spam detection, part-of-speech tagging

Deep Learning Era (2012–Present)

- Neural networks outperform traditional models
- Key breakthrough: word vectors + deep architectures
- CNNs and RNNs applied to text
- Big milestone: AlexNet (image domain), followed by LSTMs for language

Embeddings and Representation Learning

- Word2Vec (2013): learned vector representations of words
- GloVe, fastText followed
- Words mapped to high-dimensional space: similar words close together
- Limitations: static, context-independent

From GenAI to Reasoning

- GPT-3, ChatGPT, Claude, Gemini: fluent, generative capabilities
- Emergence of few-shot and zero-shot abilities
- Growing focus on chain-of-thought, tool use, planning
- Challenge: hallucinations, reasoning limitations (Examples?)

How LLMs Work

Transformer Architecture (High-Level)

- Introduced in 'Attention Is All You Need' (2017)
- Core idea: self-attention mechanism
- Processes all tokens in parallel, unlike RNNs
- Enables scalability and context-aware generation

Core Components: Self-Attention and Encoding

- Self-Attention: computes weighted relevance between all token pairs
- Tokenization: breaks text into subwords or tokens (e.g., byte-pair encoding)
- Positional Encoding: adds order info since attention is permutation-invariant

Decoder-Only vs Encoder-Decoder Models

- Decoder-only (GPT-style): optimized for text generation (left-to-right)
- Encoder-decoder (T5-style): used for translation, summarization, etc.
- Encoder-only (BERT-style): bidirectional context, good for classification

Training LLMs

- Objective: predict the next token given prior context
- Pretrained on massive web-scale corpora (code, StackOverflow, Wikipedia, etc.)
- Includes potentially insecure and biased data

Fine-Tuning and RLHF

- Supervised Fine-Tuning: curated prompts + answers
- RLHF (Reinforcement Learning from Human Feedback): reward models guide output quality
- Tradeoffs: aligns behavior with user preferences, but introduces new failure modes

LLM Capabilities

What LLMs Do Well: Code Completion

- Predictive code completion in IDEs
- Useful for boilerplate code and repetitive patterns
- Accelerates developer workflow

What LLMs Do Well: Pattern Matching

- Excellent at recognizing and repeating patterns
- Learns common idioms from training data
- Great for documentation or syntax examples

What LLMs Do Well: Reasoning (Sort of)

- Performs some reasoning in zeroand few-shot settings
- Chain-of-thought prompts improve performance
- Still limited compared to human logic

Limitations: Reasoning and Logic

- Struggles with tasks requiring logical rigor
- Prone to fallacies or inconsistent steps
- Poor handling of edge cases

Limitations: Planning and Execution

- Can’t plan across multiple steps or sessions
- Memory and context are shallow
- No true understanding of task goals

Limitations: Hallucinations and Confidence

- Confidently generates false information
- Mimics tone and style without verifying truth
- Dangerous in critical domains like law or medicine

Why LLMs 'Sound Right'

- Trained to predict likely sequences, not true ones
- Optimized for fluency, not accuracy
- Illusion of understanding due to polished output

LLMs and Code

Code Generation vs. Comprehension

- Generation: producing code snippets
- Comprehension: understanding and explaining code
- LLMs excel at one but struggle with the other

Security Risks: Vulnerable Code Patterns

- Examples: unsafe function use, hardcoded keys
- Often repeats bad practices found in public code
- Lack of risk awareness

Security Risks: Insecure Training Data

- GitHub data includes bugs and CVEs
- Models learn from both good and bad examples
- Need for curated training and fine-tuning

Why 'It Compiles' Isn’t Enough

- Compilable code can still be exploitable
- No security testing or static analysis in LLM output
- Developers may over-trust suggestions

LLM Attack Surface

Prompt Injection Basics

- User input alters model behavior via crafted prompts
- Example: override system instructions
- Exploits model's inability to distinguish intent

Jailbreaking LLMs

- Circumvents safety filters or alignment layers
- “DAN” and similar prompts to bypass content moderation
- Real-world attacks are frequent and evolving

Why Prompt Attacks Work

- Models treat input as pure context
- No real boundary between instruction and user content
- Exploitable via clever phrasing or injections

Training Data Poisoning

- Backdoors inserted via public code repositories
- Poisoned samples influence model behavior
- Can be subtle and persistent

The Model Supply Chain

- Like traditional software supply chains
- Vulnerable to upstream poisoning or manipulation
- Hard to audit or trace influence in large models

Why Attribution Matters

- Plagiarism, misinformation, and malware attribution
- Need to know what text was generated by LLMs
- Crucial for trust and accountability

Watermarking LLM Outputs

- Hidden signals embedded in generated text
- Can reveal if content came from a specific model
- Not yet standardized or widely adopted

Limits of Watermarking

- Can be stripped or defeated by paraphrasing
- Not effective in adversarial settings
- Raises privacy and forensic challenges

Red Teaming for LLMs

- Simulated adversarial testing
- Helps find vulnerabilities and jailbreak paths
- Often relies on human creativity and domain expertise

Toward Safer LLMs

- Formal evaluations and benchmarks
- Alignment techniques and continuous monitoring
- Interdisciplinary challenge: AI, UX, security, ethics