

How LLMs Process Input

Lesson 02 — AI Prompting Security

Understanding the architecture that makes prompt injection possible



Tokenization: Text to Numbers

LLMs do not see text. They see sequences of **tokens** — integer IDs from a fixed vocabulary.

```
"Hello world" → [15339, 1917]
```

```
"Ignore previous instructions" → [23182, 3766, 11470]
```

- **BPE (Byte-Pair Encoding):** the dominant tokenization algorithm
 - Common words = single token; rare words = split into sub-tokens
 - A system prompt and user input are tokenized with the **same algorithm**
 - There is no "system token type" vs. "user token type" — just integers
-

The Context Window

Everything the model sees is one flat sequence of tokens:

[SYS]	System prompt tokens
[USR]	User message tokens
[AST]	Assistant response tokens (generated)

- The labels `[SYS]`, `[USR]`, `[AST]` are special delimiter tokens, but they carry **no enforcement**
 - The model treats the entire sequence as a single input
 - Context window sizes: 4K → 8K → 128K → 1M+ tokens (growing rapidly)
-

Attention: Every Token Sees Every Token

The transformer's **self-attention mechanism** allows each token to "attend to" (look at, be influenced by) every other token in the sequence.

- A user token can attend to system prompt tokens
- The model's generated tokens are influenced by *all* preceding tokens equally
- There is no mechanism that says "attend more to system tokens"
- The model *learns* to prioritize earlier instructions, but this is a **soft tendency**, not a hard rule

Security implication: Attention treats system and user tokens identically. An injection in user tokens can influence the output just as strongly as the system prompt.

No Privilege Separation

In traditional computing:

- Kernel mode vs. user mode — hardware-enforced boundary
- Root vs. regular user — OS-enforced permissions
- SQL commands vs. parameters — prepared statement enforcement

In LLMs:

- System prompt = earlier text
 - User input = later text
 - The only "authority" the system prompt has is that it appears first
 - **No hardware, software, or mathematical enforcement exists**
-

Temperature and Sampling

After processing, the model outputs a **probability distribution** over the vocabulary for the next token.

- **Temperature = 0:** always picks the highest-probability token (deterministic)
- **Temperature = 1:** samples proportionally from the distribution (more random)
- **Temperature > 1:** flattens probabilities, increasing randomness further

For attackers: Higher temperature = more variability = sometimes the injection "slips through" even if it usually fails. Lower temperature = more predictable, but if the injection shifts the top probability, it works *every time*.

How Injections Shift Probabilities

Consider a model deciding what to output next. Without injection:

```
P("I cannot share") = 0.85  
P("The price is")    = 0.10  
P("Sure, here")      = 0.05
```

With a carefully crafted injection payload in the context:

```
P("I cannot share") = 0.30  
P("The price is")    = 0.45 ← shifted to top  
P("Sure, here")      = 0.25
```

The injection does not "override" anything — it **shifts the probability landscape** so that the compliant response becomes more likely than the refusal.

The "Instruction Hierarchy" Is Just Convention

- Models are **fine-tuned** (via RLHF, DPO, etc.) to follow system prompts
- This creates a *statistical preference*: the model usually prioritizes system-level instructions
- But "usually" is not "always" — sufficiently strong user-side context can override it
- Some providers implement a "system prompt hierarchy" in training, but it is still a learned heuristic

Key insight: Instruction-following is a *behavior learned during training*, not a *constraint enforced by the architecture*. This is the fundamental reason prompt injection cannot be "patched."

Why Fine-Tuning Helps But Doesn't Solve It

- **Safety fine-tuning** (RLHF, Constitutional AI, etc.) makes models more resistant to known attack patterns
 - But it works by shifting probabilities, not by creating hard boundaries
 - Novel attack phrasings can bypass fine-tuned defenses
 - There is an inherent cat-and-mouse dynamic: new defenses → new attacks → new defenses
 - The attacker needs to succeed **once**; the defender needs to succeed **every time**
-

Summary

- **Tokenization** converts all text to the same type of tokens — no privilege distinction
- The **context window** is a flat sequence: system + user + assistant concatenated
- **Attention** treats all tokens equally — user tokens can influence output just like system tokens
- **Temperature** affects injection reliability but does not prevent it
- Injections work by **shifting token probabilities**, not by "hacking" the model
- The instruction hierarchy is **learned, not enforced** — this is the root cause

Next lesson: Direct Prompt Injection — a comprehensive taxonomy of attack techniques.

