

DLP While Using AI

For everyone · ~10 min read

What DLP means when your "channel" is a prompt

Data Loss Prevention is a phrase that used to describe a very specific family of controls. Classic DLP watches the ways data leaves an organization — outbound email, web uploads, USB drives, chat attachments, print jobs — and either blocks, quarantines, or flags anything that matches a sensitive pattern. It is one of the oldest disciplines in enterprise security, and for two decades it worked well enough because the list of egress channels was finite and stable. Everyone knew where the doors were, and the doors had locks on them.

AI changed the shape of that problem in a single afternoon. The moment a large language model became a useful product, every employee gained a new outbound channel that the classic DLP stack cannot see: the prompt box. A prompt is an egress event. Every character you type into a chat window, every file you drop onto an assistant, every line of code your IDE quietly sends to a completion backend — each of these moves data from inside your organization to outside of it. The destination is a third party. The transport is HTTPS that looks indistinguishable from a thousand other SaaS connections. And most classic DLP rules simply do not fire on it, because they were written for email attachments and file uploads to known risky domains, not for an open-ended text field in a productivity tool.

So the definition of DLP, for the purposes of this lesson, has to be rewritten. DLP while using AI is not about watching the network. It is about a habit you perform before you hit enter: **prevent sensitive data from entering a prompt in the first place**. The only reliable control is the human in front of the keyboard deciding, every single time, whether the content in the text box is safe to send. That is an unglamorous answer, but it is the correct one, and the rest of this lesson is about making that decision fast and repeatable.

The four-class model

You cannot make a good decision if every prompt looks the same to you. The trick is to sort what you are about to send into one of four classes, in your head, before your finger hits enter. The classes are simple enough to memorize on a walk to the coffee machine, and they cover the vast majority of real-world prompts.

Class	What belongs here	Rule
Never	Secrets, credentials, API keys, private keys, session tokens, PII of real customers, health or financial data tied to a real person.	Do not paste. Ever. Not into the enterprise tier, not into a "safe" local model, not into a "just to illustrate" example.
Enterprise tier only	Source code, internal design docs, architecture and infra diagrams, ticket contents with internal names, roadmap language.	Only inside the AI tool your employer has contracted for, with zero-retention enabled, and never inside a personal account.
With redaction	Stack traces, SQL queries, log snippets, config files, error messages, curl commands.	Safe to send <i>after</i> you replace identifying values with placeholders. Keep the structure, mask the content.
Free	Public documentation, open-source code, standards text, generic conceptual questions, things you could paste into a public forum.	No restriction. This is what AI assistants are for.

The four-class model does two things at once. First, it forces you to pause and look at what you are pasting, which is half the battle — most leaks happen on autopilot. Second, it gives you a decision you can defend if anyone ever asks. "I classified it as with-redaction, and I redacted these three fields" is a coherent answer. "I felt like it was probably fine" is not.

The email test

Classifying every prompt sounds like work, and in the middle of a busy day you will want a shortcut. The shortcut is a single question you can ask yourself in about two

seconds: *would I email this to an unknown vendor?* If the answer is no, do not paste it into an AI. If the answer is yes, you are probably safe.

The test works because your brain already knows how to answer it. You have spent your entire working life deciding what is and is not appropriate to put in an email to a third party. You know instinctively that you would not email a customer's credit card number to a vendor, that you would not email production credentials to a vendor, that you would not email the full source of an internal service to a vendor you have no contract with. The AI provider is, for the purposes of this test, exactly that vendor. The fact that the interface looks like a chat window and the reply arrives in two seconds does not change the legal or operational reality — you are sending bytes to a third party. Same analysis, same answer. Running the email test out loud is often enough to catch a bad paste before it leaves.

Redaction that preserves the question

A lot of people, when they first hear the redaction rule, either over-redact (and then the model cannot help them) or under-redact (and then they have leaked anyway). Good redaction has a simple principle: **keep the shape, mask the values**. The model cares about structure, not identities. Replace the identifying content with a placeholder and leave everything else alone, and the model will still be able to answer your question.

Consider a typical environment dump someone might want to paste while debugging a deployment problem:

RAW (do not send):

```
DATABASE_URL=postgres://svc_orders:P@ssw0rd2026!@db-prod-3.internal.acme.corp
REDIS_URL=redis://:r3dis-secret-xyz@cache-prod-1.internal.acme.corp:6379
AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
SENTRY_DSN=https://abc123def456@o987654.ingest.acme.corp/1234567
FEATURE_FLAG_ENDPOINT=https://flags.internal.acme.corp/v1
LOG_LEVEL=info
ENVIRONMENT=production
```

REDACTED (safe to paste):

```
DATABASE_URL=postgres://<db_user>:<db_password>@<db_host>:5432/<db_name>
```

```
REDIS_URL=redis://:<redis_password>@<redis_host>:6379
AWS_ACCESS_KEY_ID=<aws_access_key>
AWS_SECRET_ACCESS_KEY=<aws_secret_key>
SENTRY_DSN=https://<sentry_key>@<sentry_host>/<project_id>
FEATURE_FLAG_ENDPOINT=https://<flags_host>/v1
LOG_LEVEL=info
ENVIRONMENT=production
```

Notice what the redacted version keeps and what it drops. It keeps every variable name, the protocol of every URL, the port numbers, the path structure, the log level, and the environment. All of that is what the model needs to reason about the config. It drops every real credential, every real hostname, every real project identifier. The model can still tell you that the Redis URL is using password auth, that the Sentry DSN looks structurally valid, that the database URL is pointing at a Postgres instance. It cannot, however, tell anyone else what your production passwords are, because it never saw them.

A good redaction reflex means two other things as well. First, any secret you redact must still be rotated — the fact that you replaced it with a placeholder *after* you noticed it does not undo the risk that you nearly pasted it, and you cannot be sure a copy did not end up somewhere in the meantime. Second, "expired" is not the same as "safe." A token you believe is expired might not be, session handling might be lenient, and a real token is always worth more to an attacker than a placeholder, regardless of what its expiry field claims.

Side channels

The prompt box is the obvious egress channel, but it is not the only one, and the others are easier to forget because they do not feel like typing.

- **File uploads** — dragging a PDF, CSV, DOCX, or log file onto an AI assistant is the same as pasting its contents; treat the drag as a paste.
- **Screenshots** — a screenshot of a ticket, a dashboard, or a Slack message is readable by the model and travels the same path as any other prompt.
- **Clipboard history** — many OSes and productivity tools keep a rolling clipboard buffer, and the last thing you copied may ride along with the next paste if you are not careful.

- **IDE autocomplete** — every keystroke in a completion-enabled editor is sent to a model backend, which means your entire open file is effectively one long streaming prompt.
- **Voice input** — dictation into an assistant is transcribed and sent as a prompt, so everything you say aloud near the microphone is potentially in the channel.

The "enterprise tier" myth

There is a persistent belief, especially once an organization rolls out a paid enterprise tier of an AI tool, that the enterprise tier is safe for anything. It is not. The enterprise tier is safer than a personal account along one specific axis: your prompts are contractually excluded from being used to train future models, and they are typically retained for a much shorter window, sometimes zero. That is a real and valuable improvement. It is also the entire improvement. Enterprise tier is still a vendor. Vendor admins can still see traffic during abuse investigations. Legal preservation orders still override the default retention policy. Breaches still happen to enterprise infrastructure exactly as often as to consumer infrastructure. And abuse-detection pipelines typically log prompts for some window regardless of what the retention page claims, because a vendor needs to detect misuse in real time to stay safe itself.

The only sentence to remember

Zero-retention protects you from training, not from logs, breaches, or subpoenas.

Treat the enterprise tier the way you would treat any other trusted-but-external vendor: a place where you can send things that are sensitive-to-your-company but not never-leave-the-company. Source code, internal docs, infra diagrams — yes, inside the enterprise tier. Live customer PII, production credentials, private keys, session cookies — no, not even there. The four-class model still applies.

Rules to remember

- Classify every prompt before you send it: never / enterprise-only / with-redaction / free.

- Run the email test: if you wouldn't email it to a vendor, don't paste it.
- Redact values, keep shapes — replace real IDs/secrets/hostnames with `<placeholder>`, keep the structure intact.
- Screenshots and file uploads are prompts. Treat them the same way.
- Enterprise tier protects you from training, not from logs, breaches, or subpoenas.