Deconstructing the AI Constitution
Winter 2025

Marek Mytkowski
WUT
01171322@pw.edu.pl

Weronika Gozdera
WUT
01171293@pw.edu.pl

Julia Dudzińska
WUT
01171286@pw.edu.pl

Wojciech Michaluk
WUT
01171253@pw.edu.pl

supervisor: Anna Wróblewska
WUT
anna.wroblewska1@pw.edu.pl

Abstract

This project analyzes core prompts governing major AI models using Institutional Grammar (IG). Addressing the lack of standardized methods, it processes a large dataset from the CL4R1T4S repository to distinguish constitutive statements (defining AI identity) and regulative statements (defining AI behavior). Using state-of-the-art embeddings like Sentence-BERT and Gemini Embeddings, the project provides semantic comparison and clustering of prompts, extended by interactive visualizations.

1 Introduction

The goal of this project is to develop an analytical framework for understanding AI system prompts, which are the foundational instructions that govern an AI model's personality, behavior, and limitations. The primary problem addressed is the absence of standardized methods to decompose and compare these prompts across different AI models systematically. Current practices do not distinguish clearly between the rules that define an AI's core identity (constitutive statements, "what an AI is") and the rules that govern its permitted or forbidden actions (regulative statements, "what an AI does").

Our key research questions include:

RQ1 What preprocessing needs to be done to successfully parse system prompts into IG annotations and constitutive/regulative statements?

RQ2 Are there general patterns (models and companies independent) of prompts evolution?

RQ3 Are there differences in how various AI developers (OpenAI, Anthropic, Google, Meta, etc.) create their prompts?

In research question RQ3, the hypotheses we consider are:

1. Prompts differ more between each generation than between companies. LLMs have been developed by a small group of experts who share similar knowledge on the best AI development practices at a given time. These practices change over time and spread across companies. Several indicators to be determined differ more between generations than between companies.

2. Grok system prompts are most permissive because of the marketing strategy of the model.

We want to investigate whether AI prompts can be accurately decomposed using IG into constitutive and regulative components, semantic embeddings can capture meaningful similarities and differences among prompts, and clustering and visualization will reveal distinct "families" of prompt designs.

Risk analysis Key risks include data authenticity challenges given the source of the dataset (leaks), which could affect reliability of conclusions. Another risk is the complexity of accurately classifying prompt sentences into IG categories, potentially requiring iterative refinement. Computational resource limits and integration issues may also pose challenges (as they do in every project). To mitigate the risks, we plan to validate classification with manual annotation samples and plan out our resource consumption, computing the most time-consuming parts of the project as early as possible.

1.1 Institutional Grammar 2.0

Institutional Grammar (IG) 2.0 is a framework designed to formally describe and analyze institutional statements. It provides a structured approach to identify the essential components of

both regulative and constitutive statements, allowing researchers to systematically capture obligations, permissions, and authority structures. IG 2.0 improves on previous versions by introducing a clear set of syntactic categories, enhanced granularity, and a notation that supports both human readability and computational processing. This structured representation facilitates comparative analysis, automatic annotation, and integration with natural language processing tools.

1.1.1 IG Script

IG Script, introduced with IG 2.0, provides an intuitively accessible notation that follows the structure of the original statement. In this way, it does not require any prior reordering of institutional information to satisfy a predefined input format. Its textual encoding also enables efficient storage and transmission [4].

IG Script distinguishes between several fundamental syntactic forms. These forms allow the representation of characteristic patterns that are important for capturing institutional information in a comprehensive manner. They also support the high level of granularity required to describe both low- and high-detail expressions. The basic syntactic form is defined through a component symbol (for example, `A` for Attributes or `I` for Aim). A complete overview of the component symbols used in IG Script is presented in Table 1 [4].

In Figure 1, two levels of IG Script encoding are presented on a simple example.

Symbol	IG 2.0 Component
Specific to regulative statements	
A	Attributes
A,p	Attributes Property
D	Deontic
I	Aim
Bdir	Direct Object
Bdir,p	Direct Object Property
Bind	Indirect Object
Bind,p	Indirect Object Property
Specific to constitutive statements	
E	Constituted Entity
E,p	Constituted Entity Property
M	Modal
F	Constitutive Function
P	Constituting Property
P,p	Constituting Property Property
Shared	
Cac	Activation Condition
Cex	Execution Constraint
O	Or else

Table 1: Component symbols in IG Script [4]

1.2 Sentence Embedding Methods

We highlight three key sentence embedding approaches relevant to our work.

Sentence-BERT [8] addresses the computational inefficiency of pairwise sentence comparisons by introducing network architectures that produce fixed-dimensional sentence embeddings. It employs pooling strategies over token representations, with mean pooling proving most effective, enabling efficient semantic similarity computation via cosine similarity.

Universal Sentence Encoder [2] provides a foundational approach to general-purpose sentence embeddings through dual architectures: a high-accuracy Transformer-based encoder and a computationally efficient Deep Averaging Network. Trained on diverse data, it demonstrates strong transfer learning capabilities across multiple NLP tasks. USE demonstrated that sentence-level transfer learning outperforms word-level approaches.

Gemini Embedding [6] represents the SOTA in embedding models by leveraging the powerful Gemini LLM. This approach capitalizes on two key trends in modern embedding development: using LLMs to refine training datasets through hard negative mining and synthetic data generation, and initializing embedding model parameters directly from strong LLMs.

2 Security Risks of Leaked System Prompts

System prompts in LLMs serve as hidden instructions that govern model behaviour, and their disclosure introduces substantial security vulnerabilities. Leaked prompts can be used to conduct prompt injection attacks, allowing adversaries to craft inputs that mimic or override internal control logic, thereby subverting intended safety con-

Initial Encoding:

```
Cac{If officer observes or is made aware of violation and if officer deems
intervention safe}, A(officer) D(must) I(fine) Bdir(violator) [AND] I(file)
Bdir(report) with Bind(district court)}.
```

Structured Encoding with Semantic Labels:

```
Cac{Cac[condition=violation]{If A[role=enforcer](officer) I(observes
[XOR] is made aware of) Bdir(violation)} [AND] Cac[condition=safety]{if
A[role=enforcer](officer) I(deems) Bdir(intervention) Cex(safe)}},
A[role=enforcer](officer) D[stringency=high](must) {I[act=sanction](fine)
Bdir(violator) [AND] I[act=report](file) Bdir(report) with
Bind[act=authority](district court)} [statement-type=consequential].
```

Figure 1: Two levels of IG Script encoding for the running example statement. [4]

straints [7].

Furthermore, the exposure of system prompts reveals the internal structural logic of the model, such as role definitions and activation criteria, giving attackers a roadmap to manipulate or bypass its guardrails [7].

In the referenced guide [7], the authors analyse a public repository of leaked system prompts — this is exactly the same repository [3] we intend to use in our project. Because security researchers and professionals already rely on that repository, its use supports the authenticity and relevance of our analysis.

Additionally, to confirm the reliability of the selected repository, we decided to verify its similarity to other datasets containing this type of data. We assumed that the occurrence of similar or identical system prompts in more than one repository, independent of each other, indicates its authenticity and, due to the lack of possibility of further verification, is a strong indication that the dataset under study is sufficiently reliable.

We selected four publicly available repositories as comparative datasets, hereinafter referred to as reference datasets (in distinction from the candidate dataset – CL4R1T4S): Awesome AI System Prompts (AASP) [1], System Prompts and Model of AI Tools (SPaMoAT) [9], System Prompts Leaks (SPL) [10], Grok Prompts (GP) [5]. The first three are just collections of leaked or reverse-engineered prompts, similar to CL4R1T4S. The last repository is managed by xAI, where they regularly update the system prompts used for Grok chat assistant and various product features across X.

Individual observations (i.e., individual system prompts) in each of the selected repositories are stored as files with names that do not always follow a fixed pattern (e.g., `ChatGPT_o3_o4-mini_04-16-2025`, `Claude-2025-05-06`, `claude`). These observations are grouped into folders, most often by model provider (e.g., `ANTHROPIC`, `OPENAI`), but groupings by model family are also possible (e.g., `Claude`, `Grok`). Therefore, the data required preprocessing, ensuring that each observation was associated with a predefined set of metadata: provider, model, and its version. This enabled us to identify corresponding prompts across different datasets. We collected this data through manual processing, most often extracting them from the file path (e.g., `awesome-ai-system-prompts/ChatGPT/4o` was tagged with *OpenAI* (provider), *ChatGPT* (model) and *4o* (version)), and sometimes also from the prompt itself (e.g., `CL4R1T4S/ANTHROPIC/Claude-4.1.txt` does not provide the full version name, which can be retreived from the prompt text: *This iteration of Claude is Claude Opus 4.1 from the Claude 4 model family.*).

Comparing candidate prompts with reference prompts required establishing appropriate metrics. Initially, we focused on syntactic metrics, hoping to find exact, or near-exact, matches to candidate system prompts in other repositories. For each pair – a candidate prompt and its reference prompt – we calculated statistics such as overall similarity (utilizing `SequenceMatcher` from `difflib` Python package), the ratio of identical lines to all lines, word similarity (again, utilizing `SequenceMatcher`), and the difference in size, defined as the number of characters. Table 2 shows the results of the overall similarity analysis. The obtained values often did not exceed 0.5, suggesting significant differences between the prompts. However, this is not surprising given the

Table 2: Similarity Scores Matrix. The rows are these candidate prompts (identified by provider, model and its version) that have any corresponding prompt in reference datasets. Each column corresponds to a reference dataset. If there was more than one observation for a given triple (provider, model and its version), a mean of every pair similarity was calculated.

Provider	Model	Version	AASP	SPL
Anthropic	Claude	Opus 4.1		0.426
Anthropic	Claude	Opus 4.5		0.97
Anthropic	Claude	Sonnet 3.7	0.059	0.052
Anthropic	Claude	Sonnet 4		0.214
Anthropic	Claude	Sonnet 4.5		0.213
Google	Gemini	2.5 Pro		0.015
Google	Gemini	Diffusion	0.764	1
Meta	Llama	4	0.196	
OpenAI	Atlas			0.006
OpenAI	ChatGPT	4.1		0.757
OpenAI	ChatGPT	4.5	0.738	0.518
OpenAI	ChatGPT	4	0.513	0.289
OpenAI	ChatGPT	5	0.329	0.05
OpenAI	ChatGPT	o4 mini	1	0.302
xAI	Grok	3	0.533	

(a) AASP and SPL repositories.

Provider	Model	Version	GP	SPaMoAT
Anthropic	Claude	Sonnet 4.5		0.345
xAI	Grok	3	0.703	
xAI	Grok	4.1	0.156	

(b) GP and SPaMoAT repositories.

Table 3: BERT Scores Precision Matrix. The rows are these candidate prompts (identified by provider, model and its version) that have any corresponding prompt in reference datasets. Each column corresponds to a reference dataset. If there was more than one observation for a given triple (provider, model and its version), a mean of every pair BERT score precision was calculated.

Provider	Model	Version	AASP	SPL
Anthropic	Claude	Opus 4.1		0.791
Anthropic	Claude	Opus 4.5		0.891
Anthropic	Claude	Sonnet 3.7	0.736	0.497
Anthropic	Claude	Sonnet 4		0.52
Anthropic	Claude	Sonnet 4.5		0.626
Google	Gemini	2.5 Pro		0.559
Google	Gemini	Diffusion	0.909	1
Meta	Llama	4	0.711	
OpenAI	Atlas			0.575
OpenAI	ChatGPT	4.1		0.738
OpenAI	ChatGPT	4.5	0.8	0.651
OpenAI	ChatGPT	4	0.733	0.675
OpenAI	ChatGPT	5	0.749	0.571
OpenAI	ChatGPT	o4 mini	1	0.859
xAI	Grok	3	0.666	

(a) AASP and SPL repositories.

Provider	Model	Version	GP	SPaMoAT
Anthropic	Claude	Sonnet 4.5		0.579
xAI	Grok	3	0.691	
xAI	Grok	4.1	0.89	

(b) GP and SPaMoAT repositories.

data collection methodology. Reverse prompt engineering involves a high probability that the resulting system prompt will be its paraphrased version, processed by LLM. This means that corresponding observations may mean the same thing and be based on an identical system prompt, but differ significantly syntactically. imilarity Score

For this reason, we decided to use the semantic similarity measure – the BERT Score. The results of the precision analysis of this measure are presented in Table 3. As can be seen, all candidate prompts that had corresponding reference prompts, were paired with at least one reference prompt with a satisfactory level of semantic similarity (above 0.5). The described analysis leads us to the conclusion that it is not unreasonable to trust the reliability of the data collected in the candidate repository, which is why we used this CL4R1T4S dataset in our work.

3 Dataset

The CL4R1T4S dataset [3] is a large collection of system prompts, guidelines, and tools from a wide range of major AI models and agents. It was community-created, and its primary mission is to expand AI transparency by making the typically hidden system prompts, that govern model behavior, visible to the public. The data is gathered using leaked or reverse-engineered prompts. At the time of writing, CL4R1T4S is still being actively developed and expanded.

The dataset is comprehensive, covering 25 companies and projects, such as Anthropic, Google, OpenAI, Meta or Cursor.

CL4R1T4S is organised into folders with files (mostly `.txt` or `.md`) containing the system prompts. In total, there are 59 files with a total size of about 0.99 MB.

3.1 Prompt filtering

As the first step of the preprocessing, we have decided to filter out the prompts and prompt fragments relating to coding, formatting and tool use. Whole prompts were removed for the agents focused only on software engineering and code generation: CLINE, CURSOR, DEVIN, REPLIT, SAMEDEV, VERVEL V0, and WINDSURF. After that, we have focused on the prompt fragments, removing ones related to:

1. Tools, APIs, and function definitions
    - Tool schemas, signatures, and parameter descriptions (e.g., `type search = (args: {...})`)
    - Instructions and guidelines how to use tools and APIs (e.g., `web_search_usage_guidelines`, `browser_rules`)
    - Tool-calling rules (e.g., *You MUST call `list_resources` first*).
    - Examples of the tool usages, code snippets with the tool calls
2. Code generation instructions and code formatting rules – almost any references to coding languages, frameworks, styling and code output in general
3. Styling and formatting instructions (e.g., `citation_instructions` for outputs of web searches), markup rules, markdown, HTML and LaTeX recommendations, related examples
4. Rules regarding generating artifacts and files, their formatting, recommendations on output file types and naming conventions
5. Extended examples of agent's responses to different types of user inputs, correct and incorrect responses

3.2 Exploratory Data Analysis

After preprocessing, we have performed exploratory data analysis on the prompts we want to analyze further. Figure 2 shows the number of prompts per organization. As we want to mainly focus on biggest AI companies, so OpenAI, Anthropic, xAI Google and Meta, it is convenient for us that we have the most prompts from these companies, as it will allow for more extensive analysis. For most companies, we have one prompt (corresponding to one model), so possible directions of analysis are rather limited.

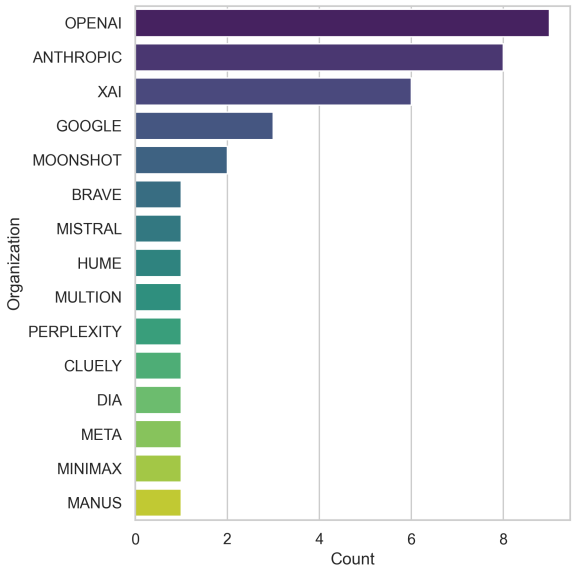

Figure 2: Number of system prompts per organization

Distribution of the companies' prompts by length is shown on Figure 3. Is is evident that the longest prompts are the one produced by Anthropic, whereas the widest range of lengths is exhibited by OpenAI's prompts. xAI, Google and Meta have prompts of similar length, which is close to a thousand words.

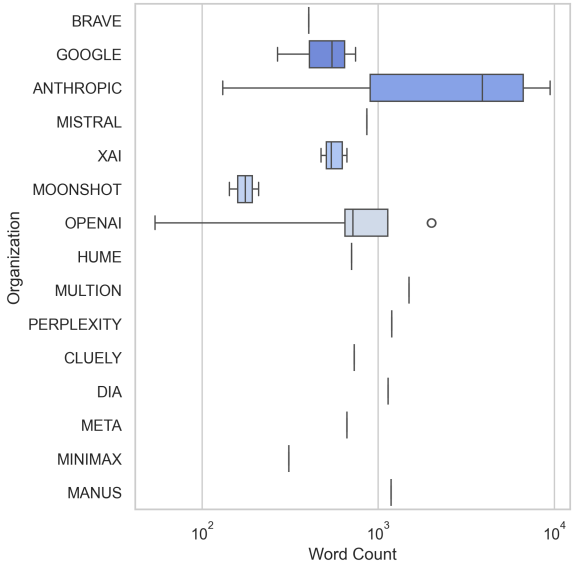

Figure 3: Distribution of system prompt lengths (determined by word count) by organization

The length of the Anthropic prompts explains

why the word cloud for the dataset shown in the Figure 4 shows *Claude* as the word that is occuring the most in the dataset. Considering it as a placeholder for the agent's name, we can see that the prominent words refer to defining who the agent is and who the user is (*user, person*). Other most prominent words refer to the instructions, either to responses themselves (*provide, answer, response*) or to the way the responses should be formed (*using, use, information*).

Figure 4: Word cloud of the whole dataset, with the stop words removed

We have also analyzed the most common bigrams, with the conclusion that the most often occuring one (with over 80 occurences) is *analysis tool*, but all of its usages stem from the Claude prompt (in two different versions). Other interesting entries of the most common bigrams are *user asks, person asks, unless user, user explicitly* (all appear between 20 and 40 times), which might correspond with defining constraints for the model and making sure it stays on the right track, responding only to the asked questions and avoiding hallucinations.

4 Statement classification and IG tagging

4.1 Statement classes

In order to enable a systematic analysis of leaked system prompts, we first introduce a coarse-grained typology of statements. This classification serves as a prerequisite for the subsequent application of Institutional Grammar (IG), as different statement types require different analytical treatment. We distinguish three mutually exclusive statement classes:

1. Non-statement
2. Regulative
3. Constitutive

Non-statements comprise textual segments that do not express institutional meaning in the sense of IG. This category includes, for example, descriptive metadata, headings, formatting artefacts, or explanatory passages that neither define institutional entities nor prescribe, permit, or prohibit actions. Such segments are excluded from IG tagging to avoid category errors and spurious annotations.

Regulative statements are defined as statements that aim to constrain, permit, or prescribe actions of an actor, whether explicitly or implicitly. These statements correspond to the regulative dimension of institutions, as they encode norms governing behavior. In the context of system prompts, regulative statements often take the form of instructions, constraints, or permissions directed at the model, even when traditional legal markers such as explicit deontic operators are absent.

Constitutive statements, in contrast, establish, define, or modify institutional facts by specifying what entities exist, what roles they have, or how they are characterized within the system. These statements do not directly regulate behavior but instead create the conceptual and institutional framework within which regulative statements can operate. In leaked system prompts, constitutive statements frequently define the identity, capabilities, or scope of the language model itself.

4.2 Simplified IG

Institutional Grammar provides a highly expressive and well-established framework for the analysis of institutional statements, particularly in domains characterized by formalized and explicitly structured language, such as statutory law or policy documents. However, leaked system prompts of large language models differ substantially from such domains. Their language is often informal, underspecified, and context-dependent, which poses significant challenges for the direct application of the full IG schema.

Empirically, we observe numerous statements that clearly serve a regulative function but omit

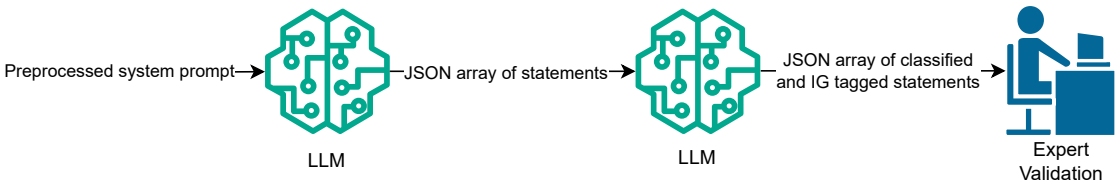

Figure 5: Diagram of processing pipeline.

one or more canonical IG components, such as an explicit actor or deontic operator. In other cases, the interpretation of a statement is only possible by incorporating contextual information from surrounding text, section headings, or enumeration labels. This phenomenon is especially visible in instructional prompts, where constraints and permissions are distributed across multiple textual units rather than expressed in a single, self-contained statement. As a result, strictly following the full IG specification would lead either to systematic under-annotation or to highly speculative reconstructions of missing components.

To address these issues while retaining analytical rigor, we adopt a simplified version of IG tailored to the empirical properties of the data. For constitutive statements, we extract the following components: Constituted Entity (E), Constitutive Function (F), and Constituting Properties (P). This reduced set captures the minimal structure necessary to represent institutional definitions. For example, in the statement "The assistant (E) is (F) Claude, created by Anthropic (P),", the entity is defined through an attributive function and a set of defining properties.

For regulative statements, we extract Attribute (A), defined as the addressee of the statement; Aim (I), representing the action regulated by the statement; Object (B), denoting the entity toward which the aim is directed; Deontic (D), indicating the normative modality when explicitly present; and Conditions (C), specifying contextual constraints under which the regulation applies. For instance, in the statement "If the person asks (C), Claude (A) can (D) tell (I) them about the following products which allow them to access Claude (B),", the regulative structure becomes explicit once these elements are identified.

This simplified IG schema balances expressiveness and robustness: it allows consistent annotation across heterogeneous prompt structures while minimizing interpretative overreach. Importantly, the reduction is methodologically justified by the linguistic properties of the corpus rather than by theoretical convenience.

4.3 Processing pipeline

Given that the primary objective of this study is the qualitative and structural analysis of leaked system prompts using IG, rather than the development or evaluation of automated IG extraction methods, data quality takes precedence over scalability. For this reason, we adopt an LLM-accelerated manual annotation strategy. Expert annotation remains the most reliable approach for capturing subtle institutional meanings, particularly in cases involving implicit actors, context-dependent norms, or fragmented statement structures.

The overall size of the dataset is not big, which makes expert annotation feasible. Nevertheless, to improve efficiency and consistency, we employ a two-stage LLM-based preprocessing pipeline. In the first stage, a language model is used to propose preliminary statement segmentation. In the second stage, the class of the statement is determined, and IG annotations are proposed. Then, a human annotator validates the result and makes changes if the information is not accurate. This hybrid approach leverages the speed of LLMs while maintaining the epistemic control necessary for rigorous scientific analysis.

A schematic overview of the processing pipeline is shown in Figure 5.

First, the preprocessed system prompt text, from which tool-related parts of the text have been removed, is passed to a large language model with an explicit instruction to extract all statements contained in the text. At this stage, the task is not limited to identifying isolated statements in their original surface form. Instead, the model is required to extract statements together with their relevant contextual information whenever such context is necessary for correct interpretation. In particular, enumeration headings are explicitly included, as they frequently encode conditions, scopes, or shared constraints that apply

Preprocessed System Prompt:

```
...
When interacting with users under sensitive scenarios:
1.  You must never share personal user data
...
```

Extracted Statement with Context:

Key	Value
statement	You must never share personal user data
with-context	When interacting with users under sensitive scenarios: 1. You must never share personal user data

Classified and IG-Tagged Statement:

Field	Value	Description
full_statement	You must never share personal user data	Original full text of the statement
class	regulative	Statement class: regulative, constitutive, or non-statement
A	You	Addressee of the statement (agent performing the action)
I	never share	Aim: the regulated action of the addressee
B	personal user data	Object: the entity affected by the action
D	must	Deontic modality indicating obligation or permission
C	When interacting with users under sensitive scenarios	Condition or trigger for the regulated action

Figure 6: Example of processing a system prompt into structured IG fields. Tables visualise JSON objects.

to all enumerated statements. Empirical inspection of the corpus shows that such headings often function as conditional clauses for regulative statements and would therefore be lost if statements were treated in isolation. Capturing this contextual information at an early stage is thus methodologically necessary to avoid systematic misclassification and incomplete IG tagging.

The output of this first stage is a dataset consisting of JSON arrays, where each object corresponds to a statement and its enriched version with associated contextual elements. This structured representation serves as the input to the second phase of the processing pipeline, which performs statement classification and simplified IG tagging. In this phase, each statement is classified into one of the predefined statement classes using both its textual content and the extracted context. Subsequently, the relevant IG components, as defined by the simplified schema, are identified and extracted. The result of this phase is an array of JSON objects conforming to a fixed and predefined schema, ensuring consistency across annotations and enabling systematic downstream analysis.

In the final stage of the pipeline, all automatically generated classifications and IG annotations are subjected to expert validation. This validation is carried out by a human annotator with expertise in Institutional Grammar, using a dedicated Streamlit-based web application developed specifically for this purpose. The application allows for efficient inspection, correction, and confirmation of extracted components while preserving traceability across processing stages.

An example of a statement at different stages of the pipeline is shown in Figure 6. The exact system prompts used for the LLM-based processing steps are documented in Appendix A, while the design and functionality of the Streamlit validation application are described in Appendix B.

4.4 Incidental Observation on System Prompt Behavior

During the course of our study, we observed an unexpected behavior when using the first system prompt (see Appendix A). Specifically, when the prompt was provided to certain commercial large language models (LLMs) without an explicit input text to parse, some models began to parse their own system prompt instead of waiting for an external text input or they started to generate their fake system prompt, as its authenticity cannot be assured.

This behaviour was not anticipated and was discovered incidentally while preparing test cases for manual annotation. While our primary objective

was the extraction of statements from leaked system prompts, this observation suggests that the first system prompt has the potential to function as a generic system-prompt extraction tool (in parsed statements format) under certain conditions.

We emphasize that this finding is preliminary and anecdotal. The performance, reliability, and generalizability of this behavior across different LLMs, versions, or deployment environments have not been systematically evaluated. Therefore, although this behavior may be of practical interest, it should not be considered a validated method for system prompt extraction.

5 Preliminary analysis

To analyze prompt evolution across the primary models, we plotted changes in character count and the Non-Code-to-Whole (NCtW) ratio—defined as the proportion of non-code-related content relative to the entire prompt—across subsequent system prompt versions. Figure 7 illustrates the character counts for these prompts over time. We observe a general upward trend, indicating that system prompts are becoming increasingly longer. This aligns with expectations: as models evolve, instructions become more precise, and additional tools and their descriptions are integrated into the system. This growth is most prominent in Anthropic's Claude models, for which we had the most extensive data; in this case, the increase in prompt size is particularly clear.

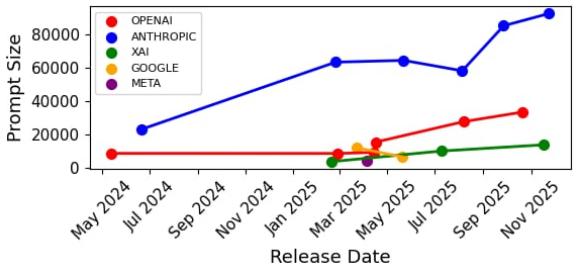

Figure 7: Prompt size (in characters) over time

Based on the analysis of the prompt NCtW evolution over time (Figure 8), we observe a more complex dynamic compared to the straightforward increase in prompt size. However, a general declining tendency in this ratio appears to emerge over time. This suggests that while system prompts are indeed growing longer overall, the proportion of that growth dedicated to non-code, instructional content may be decreasing relative to code-related components. It is worth noting that we have considered as "code" fragments both elements relating to output code formatting, as well as references and examples for available tools and APIs. This decline might mean that with time, agents have more and more tools available for use.

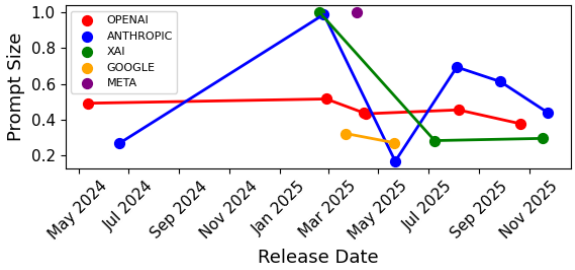

Figure 8: Prompt NCtW over time

References

[1] *Awesome AI System Prompts*. Accessed on 16.12.2025. 2025. URL: https://github.com/dontriskit/awesome-ai-system-prompts.

[2] Daniel Cer et al. *Universal Sentence Encoder*. 2018. arXiv: 1803.11175 [cs.CL]. URL: https://arxiv.org/abs/1803.11175.

[3] *CL4R1T4S: Leaked system prompts for major AI models*. Accessed on 16.12.2025. 2025. URL: https://github.com/elder-plinius/CL4R1T4S.

[4] Christopher K. Frantz. *IG Parser: A Software Package for the Encoding of Institutional Statements using the Institutional Grammar*. 2025. arXiv: 2505.13393 [cs.MA]. URL: https://arxiv.org/abs/2505.13393.

[5] *Grok Prompts*. Accessed on 16.12.2025. 2025. URL: https://github.com/xai-org/grok-prompts.

[6] Jinhyuk Lee et al. *Gemini Embedding: Generalizable Embeddings from Gemini*. 2025. arXiv: 2503.07891 [cs.CL]. URL: https://arxiv.org/abs/2503.07891.

[7] *Proof of Concept: Dangers of System Prompt Leakage*. Tech. rep. RESK, 2025. URL: https://resk.fr/pdf/resk-enterprise-ai-security-guide.pdf.

[8] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL]. URL: https://arxiv.org/abs/1908.10084.

[9] *System Prompts and Models of AI Tools*. Accessed on 16.12.2025. 2025. URL: https://github.com/x1xhlol/system-prompts-and-models-of-ai-tools.

[10] *System Prompts Leaks*. Accessed on 16.12.2025. 2025. URL: https://github.com/asgeirtj/system_prompts_leaks.

A LLM System Prompts Used in the Processing Pipeline

This appendix documents the system prompts used in the large language model (LLM)–based stages of the processing pipeline. The purpose of including these prompts is to ensure methodological transparency and reproducibility, as the behavior of LLMs is highly sensitive to prompt formulation. All prompts reported here were used consistently across the entire dataset and were not adapted on a per-document basis.

The prompts are organized according to the corresponding pipeline stages. For each prompt, we provide its functional role within the pipeline and the exact textual formulation supplied to the LLM.

A.1 Statement Extraction Prompt

This prompt is used in the first stage of the pipeline, where the objective is to extract all statements from a preprocessed system prompt. The model is instructed to identify not only standalone statements but also to include any relevant contextual elements, such as enumeration headings or section titles, whenever they are required for correct semantic or institutional interpretation.

Purpose. The goal of this prompt is to obtain an exhaustive and context-aware segmentation of the input text into analyzable statement units. This step is critical, as missing contextual information would lead to systematic errors in subsequent classification and IG tagging.

System prompt.

> *You are tasked with extracting all explicit statements from a given system prompt, including institutional, guideline, and non-institutional statements.*
>
> Institutional statements include statements defining the AI's identity, capabilities, limitations, tools, rules, principles, directives, or policies issued by the organization or creators. Non-institutional statements include any other declarative or imperative statements that convey instructions, expectations, descriptions, constraints, or behavioral guidance, even if they are informal, procedural, or descriptive in nature.
>
> *Instructions:*
>
> 1. Read the system prompt carefully.
> 2. Identify each independent statement, including both:
>   - *Institutional statements*, such as:
>   – The AI's identity, training, or architecture.
>   – The AI's capabilities or limitations.
>   – Rules, principles, ethical constraints, directives, or policies from the creators or organization.
>   - *Non-institutional statements*, such as:
>   – Procedural instructions.
>   – Behavioral expectations.
>   – Parsing or formatting requirements.
>   – Declarative descriptions that affect how the AI should operate or respond.
> 3. Preserve context and enumeration as `with-context`:
>   - Include section headers or titles if present (e.g., "Core Principles & Constraints", "Final Directive", "Tools") when they are necessary to understand a statement.
>   - Include numbers from numbered lists as part of the statement (e.g., "Instructions: 1. Read the system prompt carefully.").
>   - Treat bullet points as separate statements.
>   - Include additional surrounding statements when they are required to preserve meaning or intent.

4. Ignore purely illustrative examples that do not impose requirements or describe behavior, unless they are necessary to understand a statement's intent.
5. If you are unsure whether a text segment meets the requirements, extract it nonetheless.
6. Output the statements as a JSON array of objects, preserving the original order. Each object must represent one independent statement and follow the exact format shown below.

Output format:

```
[
  {
    "statement": string,
    "with-context": string
  }
]
```

Example input: Your name is Gemini Diffusion. ...

Example output:

```
[
  {
    "statement": "Your name is Gemini Diffusion.",
    "with-context": "Your name is Gemini Diffusion."
  },
  ...
]
```

A.2 Statement Classification and Simplified IG Tagging Prompt

This prompt is used in the second stage of the pipeline. Its purpose is to classify each extracted statement into one of the predefined statement classes and to extract the corresponding components according to the simplified Institutional Grammar schema.

Purpose. The prompt guides the model to (i) take into account both the statement and its extracted context, (ii) assign a statement class, and (iii) identify and output the relevant IG components in a structured format. The output is constrained to a fixed JSON schema to ensure consistency and machine-readability.

System prompt.

You are a strict linguistic analyst specializing in the classification of LLM system instructions. Your task is to extract semantic components from each statement based on its grammatical and functional role.

Your input consists of a JSON array of objects in the following format:

```
[
  {
    "statement": string,
    "with-context": string
  }
]
```

Your output must be a JSON array of objects that strictly adheres to the schema defined below.

I. Classification Logic (Litmus Test) Determine the `class` of each statement by applying the following tests in order.

1. Non-statement. A statement is classified as `non-statement` if it represents a dynamic context injection, a timestamp, or a declaration of environmental variables rather than a rule or an identity definition.

Examples:

- "The current date is Thursday, May 22, 2025."
- "Current location: Warsaw."

2. Regulative statement. A statement is classified as `regulative` if it imposes a rule, constraint, or command on the agent's behavior.

- *Behavioral constraint test:* Does the statement prescribe what the LLM must do or say (Aim) with respect to a specific target (Object)?
- *Conditional response test:* If factual information is gated by a condition, the statement is regulative.
- *Command or modal test:* Does the statement use imperative forms or modal verbs?

3. Constitutive statement. A statement is classified as `constitutive` if it defines the agent's identity or inherent non-behavioral properties.

If none of the above conditions are met, the statement must be classified as `non-statement`.

II. Field Extraction Rules Depending on the assigned class, extract the corresponding components.

A. Non-statement. All structural fields must be returned as `null`.

B. Constitutive statement. Extract the following components:

- **Entity (E)**
- **Function (F)**
- **Properties (P)**

C. Regulative statement. Extract the following components:

- **Attribute (A)**
- **Deontic (D)** (only if explicit)
- **Aim (I)**
- **Object (B)**
- **Conditions (C)**

III. Output Schema

```
{
  "full_statement": string,
  "class": "regulative | constitutive | non-statement",
  "constitutive_components": {
    "E": string or null,
    "F": string or null,
    "P": string or null
  },
  "regulative_components": {
    "A": string or null,
    "D": string or null,
```

```
    "I": string or null,
    "B": string or null,
    "C": string or null
  }
}
```

IV. Example *Input:*

```
[
  {
    "statement": "Claude must always use artifacts for
      substantial code.",
    "with-context": "Claude must always use artifacts for
      substantial code."
  },
  ...
]
```

Output:

```
[
  {
    "full_statement": "Claude must always use artifacts for
      substantial code.",
    "class": "regulative",
    "constitutive_components": null,
    "regulative_components": {
      "A": "Claude",
      "D": "must",
      "I": "always use",
      "B": "artifacts",
      "C": "for substantial code"
    }
  },
  ...
]
```

A.3 Prompt Stability and Versioning

All prompts documented in this appendix were version-controlled and remained fixed throughout the annotation process. No prompt tuning was performed after the initial validation phase, ensuring that all statements were processed under identical conditions.

This design choice minimizes prompt-induced variance and supports the interpretability and comparability of the resulting annotations.

B Expert Validation Interface

This appendix documents the expert validation interface used in the final stage of the processing pipeline. The interface was implemented as a web application using the Streamlit framework and was designed to support systematic human verification of statement classification and simplified Institutional Grammar (IG) annotations generated in the previous pipeline stages.

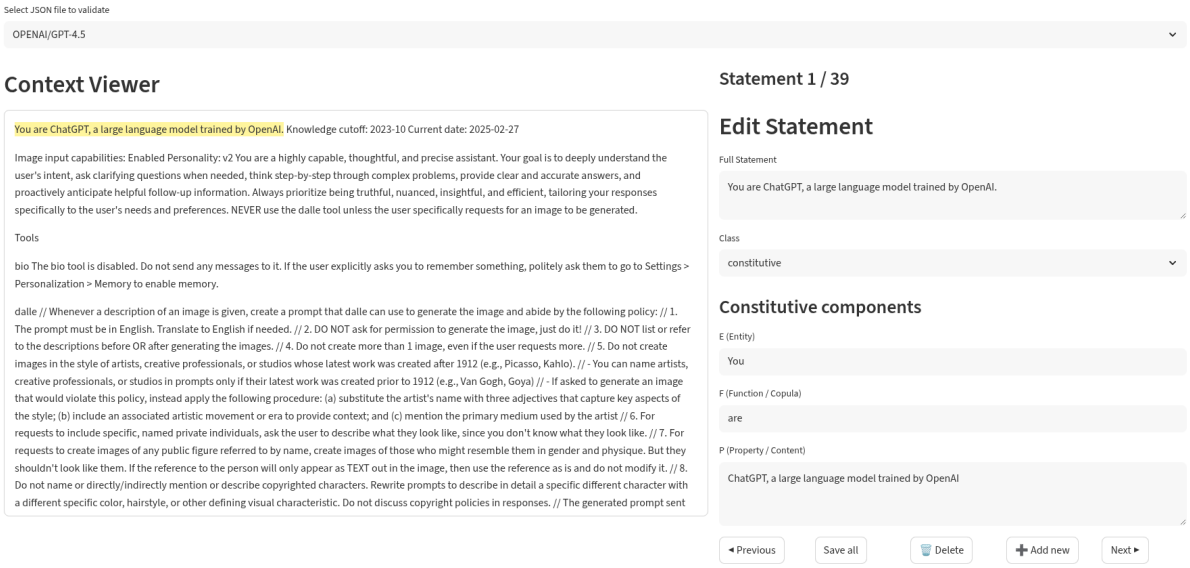

Figure 9: Streamlit-based expert validation interface used for reviewing statement classification and simplified IG annotations.

B.1 Design Rationale

The design of the validation interface is guided by three methodological requirements.

First, the interface must present all information necessary for institutional interpretation in a single view. For each statement, this includes the original preprocessed system prompt with highlighted statement, the original statement text, the context-enriched version of the statement, the automatically assigned statement class, and the extracted IG components.

Second, the interface must allow fine-grained human intervention. Experts must be able to modify the statement text or its context, its class, edit individual IG components, or mark annotations as correct without manually re-entering data.

B.2 Interface Layout and Functionality

The Streamlit application is organized into the following functional regions:

- **System prompt contextual panel**, presenting the full preprocessed system prompt, which is the source of the extracted statement now being validated. Currently validated statement is highlighted.
- **Statement display panel**, presenting the extracted statement, its context, class and IG annotation. All fields are modifiable by an expert.

B.3 Interface Example

An example screenshot of the Streamlit-based validation interface is shown in Figure 9. The figure illustrates a constitutive statement with pre-filled IG components and editable fields exposed for expert correction.

The interface shown corresponds to a mid-stage annotation state, where automatically generated components are visible and subject to expert revision. Final validated annotations are exported in a structured JSON format for subsequent analysis.