Università degli Studi di Milano Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Dottorato in Informatica

Refining Large Language Models Outputs for Legal Information Extraction

Neuro-Symbolic Integration Course

Student: Renzo Arturo Alva Principe

ID: Matricola 746799

Cycle: XXXVII

Grammar-Constrained Decoding for Structured NLP Tasks without Fine Tuning

Text Generation in Predefined Format

Context:

- LLMs have become very popular as achieved impressive results in many NLP tasks
- they can be adapted or via fine-tuning or via in-context learning prompting
- Certain tasks require the output to follow a predefined format

Problem

- LLMs still struggle with reliably generating complex output structures
- fine tuning is an option but require large datasets
- constrained generation: approaches use fine-state machines as representation model

Idea: use grammars as a language to encode output formats and integrate with LLMs to enforce generation

Why Grammars?

- They show that for many NLP tasks the respective output space can be described with a formal grammar
- focus in writing grammars allows to ignore the implementation details
- wrt finite-state automatas of tree-based representations grammars require less effort

Token-Level Grammars

A formal grammar G is defined as a tuple (V, Σ,P,S):

- V is a finite set of non-terminal symbols
- Σ is a finite set of terminal symbols
- P is a finite set of production rules
- S ∈ V is the start symbol.

- the user first writes a formal grammar G over characters
- we need a token-level grammar G_{tok} that can be used to constraint the LLM
- they build the token-level grammar G_{tok}= (V, Σ_{tok},P_{tok},S) by applying the tokenizer
- they then use an incremental parser to decide whether a token sequence y is in the language generated by G_{tok}

Grammar-Constrained Decoding

- The LLM decoding process produces tokens one by one
- during decoding probability distribution of generated tokens is pruned to include only the subset of tokens that are allowed by the formal grammar

Use Case

Context: traffic violation appeals

Traffic code violation leads usually to a *fine*

The accused can submit an *appeal document* to the *Prefecture*

In the appeal document the accused requests to an annulment or a reduction of the penalty

Problem: streamline the backoffice

Appeals need to be managed: the Prefecture and the Giudice di Pace are designed for this work

Appeals *are not all the same*.
They can vary in the: motivations, evidence provided, requests, violated articles, information provided

Each case is assigned to a lawyer; however, some cases require a high level of expertise from the lawyers.

The municipality of Naples has **70,000 appeals to review**…

Commissioner Task: Information Extraction

The Prefecture of Naples requires to:

(1) Classify all the documentation sent by the applicant
(2) Extract the relevant information ← **focus of the presentation**
(3) Classify the cases based on difficulty to assign them to lawyers according to their experience.

Subtasks identified:

Entity Extraction & **Relation Extraction**

The Appeal: structure and information

Document structure:

- **Header**: recipient, and subject.
- **Appeal Against**: details the specific violation contested.
- **Applicant's Information**: personal details of the appellant.
- **Premise**: background information and context of the violation.
- **Argument**: rights and justifications for contesting the violation.
- **Requests**: specific requests for annulment or action.
- **Attachments**: supporting documents.

Entities to extract:

- verbale numbers
- license plate
- dates
- CF lawyers
- CF applicant
- recipient
- legal mail
- time stamps
- registry code

Relation to extract:

- date of violation: verbal → date
- date of notification: verbal → date

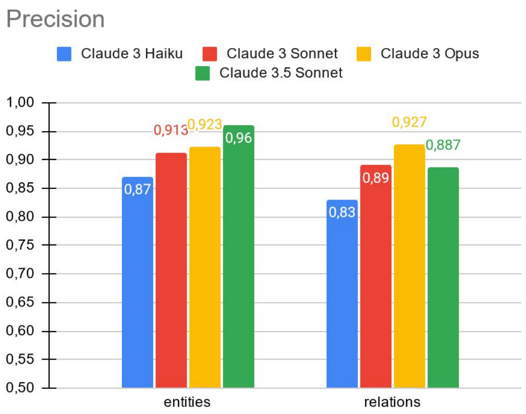
Dataset and Output Format

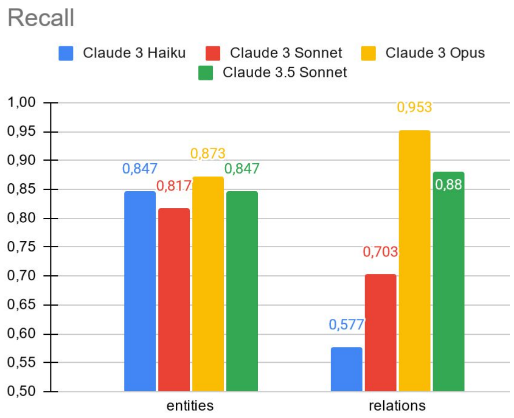
- Data manually annotated
- Development set: 8 instances
- Test set: 15 instances

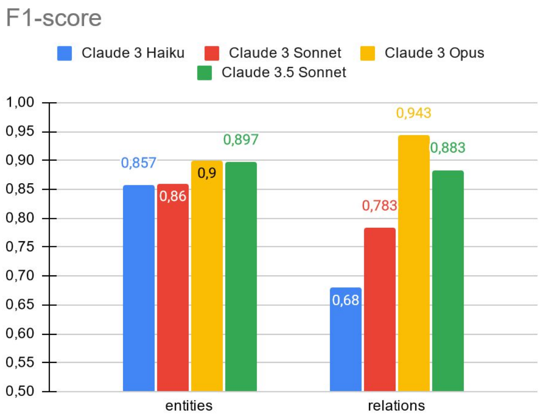

```
{
    "entities":{
        "PREFETTO":"destinatario",
        "xxxx":"cf_trasgressore",
        "yyyy":"cf_avvocato",
        "xxx@legalmail.it":"mail",
        "AP34534435334":"num_verbale",
        "ZZZZ":"targa",
        "09/09/57":"data",
        "27/08/2022":"data",
        "29/05/2022":"data",
        "orario":"00:10",
        "orario":"10:35",
        "num_registro":"24542554"
    },
    "relations":[
        {
            "source":"AP34534435334",
            "relation":"data_notifica",
            "target":"27/08/2022"
        },
        {
            "source":"AP34534435334",
            "relation":"data_infrazione",
            "target":"29/05/2022"
        }
    ]
}
```

What has been done so far

Closed-source Large Language Models perform good!







but…

Business Task: Can we reduce the cost?

Claude 3 Opus costs averagely 0,032$* for each document processed

$\longrightarrow$ the 70k appeals cost 2240$*

Appeals increase each year and we are only considering Naples…

Can we reduce the Information Extraction cost?

* the whole set of tasks costs 0,054$ per document $\longrightarrow$ 4060$

Open Source LLMs

- Open source LLMs are usually less expensive
- on-premise deployments contributes to keep costs low
- LLama 3.1 8B is both powerful and compact ⟶ our choice

However….

- performance are lower than closed-source solutions
- the output of IE is usually a JSON ⟶ Llama has difficult hadering structured output

Idea:

- **Fine-Tuning** should enhance performance
- **Logits Constraint** should help enforce the correct output format

Fine Tuning

- LoRA adapters were used to avoid the full-fine tuning of the LLM
- However even with this facilitation we need to use quantization of the LLM weights in order to reduce memory consumption → from 32 bit to 4 bit
- We reduce context window from 120k to 12k to reduce computation costs but still fitting our data

Logit Constraints

Grammar for a valid JSON:

```
root   ::= object
object ::= "{" ws ( string ":" ws value ("," ws string ":" ws value)* )? "}"
value  ::= object | array | string | number | ("true" | "false" | "null") ws
array  ::= "[" ws ( value ("," ws value)* )? "]" ws
string ::= "\"" [ \t!#-\[\]-~]* "\"" ws
number ::= ("-"? ([0-9] | [1-9] [0-9]*)) ("." [0-9]+)? ([eE] [-+]? [0-9]+)? ws
ws ::= ([ \t\n] ws)?
```

Grammar for a valid IE JSON output:

```
root   ::= "{" ws "\"entities\"" ":" ws object "," ws  "\"relations\"" ":" ws array "}"
object ::= "{" ws ( string ":" ws value ("," ws string ":" ws value)* )? "}"
value  ::= object | string | number  ws
string ::= "\"" [ \t!#-\[\]-~]* "\"" ws
number ::= ("-"? ([0-9] | [1-9] [0-9]*)) ("." [0-9]+)? ([eE] [-+]? [0-9]+)? ws
array  ::= "[" ws ( object ("," ws object)* )? "]" ws
ws ::= ([ \t\n] ws)?
```

Constraint: None Vs JSON Vs Custom (One-shot)

	Fine Tuning	Constraints	IE Performance			RE Performance			Errors
LLM	max_steps	type	F1	Precision	Recall	F1	Precision	Recall	Malformed JSON
LLama 3.1 8B 4bit	20	-	0,05	0,875	0,02	0	0	0	14
LLama 3.1 8B 4bit	20	JSON	0,24	0,66	0,14	0,03	0,08	0,02	6
LLama 3.1 8B 4bit	20	JSON custom	**0,24**	0,66	0,14	**0,03**	0,08	0,02	**6**
LLama 3.1 8B 4bit	60	-	0	0	0	0	0	0	15
LLama 3.1 8B 4bit	60	JSON	0,12	0,22	0,08	0	0	0	7
LLama 3.1 8B 4bit	60	JSON custom	**0,.14**	0,2	0,11	0	0	0	**7**
LLama 3.1 8B 4bit	100	-	0,15	0,6	0,08	0	0	0	12
LLama 3.1 8B 4bit	100	JSON	0,42	0,43	0,41	0,04	1	0,02	3
LLama 3.1 8B 4bit	100	JSON custom	**0,45**	0,45	0,44	**0,06**	0,75	0,03	**2**

	Fine Tuning	Constraints	IE Performance			RE Performance			Errors
LLM	max_steps	type	F1	Precision	Recall	F1	Precision	Recall	Malformed JSON
LLama 3.1 8B Instruct 4bit	20	-	**0,49**	0,7	0,37	**0,34**	0,95	0,21	**6**
LLama 3.1 8B Instruct 4bit	20	JSON	0,18	0,65	0,1	0,32	0,9	0,2	12
LLama 3.1 8B Instruct 4bit	20	JSON custom	0,18	0,65	0,1	0,32	0,9	0,2	12
LLama 3.1 8B Instruct 4bit	60	-	**0,56**	0,56	0,56	**0,56**	0,59	0,54	**0**
LLama 3.1 8B Instruct 4bit	60	JSON	0,07	0,9	0,03	0	0	0	14
LLama 3.1 8B Instruct 4bit	60	JSON custom	0,09	0,48	0,09	0	0	0	13
LLama 3.1 8B Instruct 4bit	100	-	**0,58**	0,62	0,55	**0,67**	0,71	0,64	**1**
LLama 3.1 8B Instruct 4bit	100	JSON	0,19	1.11	0,11	0,14	1	0,07	13
LLama 3.1 8B Instruct 4bit	100	JSON custom	0,21	0,51	0,13	0,29	0,94	0,17	12

Discussion

Fine tuning:

- had a little effect on Llama 3.1 but while in Llama 3.1 Instruct the effect is visible
- it helps to reduce malformed errors especially in the Instruct version

Logit Constraints:

- in Llama 3.1 the effect was impressive both in performance and malformed errors
- in the Instruct version the effect was devastating introducing also malformation errors

Base models

	Constraints	IE Performance			RE Performance			Errors
	type	F1	Precision	Recall	F1	Precision	Recall	Malformed JSON
LLama 3.1 8B 4bit	-	**0,39**	0,77	0,26	**0,14**	0,21	0,11	**5**
LLama 3.1 8B 4bit	base	0,05	0,88	0,03	-	-	0	11
LLama 3.1 8B 4bit	JSON	0,05	0,88	0,03	-	-	0	14
LLama 3.1 8B Instruct 4bit	-	**0,61**	0,79	0,5	**0,67**	0,8	0,57	**1**
LLama 3.1 8B Instruct 4bit	base	0,23	0,97	0,13	0,36	0,95	0,22	12
LLama 3.1 8B Instruct 4bit	JSON	0,23	0,97	0,13	0,36	0,95	0,22	10

Discussion:

- Constraints alone have a very bad influence on not fine tuned models
- However considering LLama 3.1 the mix of constraints with fine tuning outperform the base model performance F1 (0,45 Vs 0,39)
- Instruct is better if not touched

Conclusion

- constraints seem to have only a good influence when the model on which are applied perform very poorly for example on LLama 3.1
- constraints seem have a bad effect on any other cases
- fine tuning seems to work only with Instruct, however keep the model untouched is still better
- However when used with fine tuning can led to high performance
- We are still far from F1 performance of top LLMs :
    - entities: 0,61 (best Llama 3.1 Instruct)   Vs 0,9 (Claude 3 Opus)
    - relations 0,67 (best Llama 3.1 Instruct)   Vs 0,94 (Claude 3 Opus)

⟶ the problem with fine tuning may be both a problem of quantization and very poor data

⟶ the problem of constraints may be an knownproblem with long inputs