CodeIE: Large Code Generation Models are Better Few-Shot Information Extractors

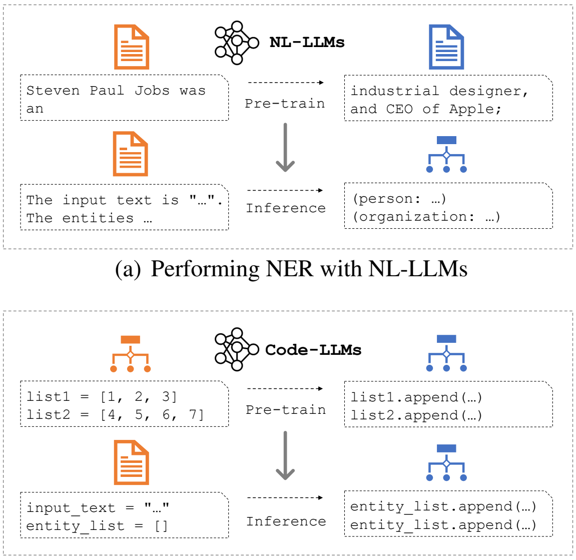
Peng Li[1,*] **Tianxiang Sun**[2,*] **Qiong Tang**[2] **Hang Yan**[2]
Yuanbin Wu[3] **Xuanjing Huang**[2] **Xipeng Qiu**[2,†]
[1]Academy for Engineering & Technology, Fudan University
[2]School of Computer Science, Fudan University
[3]School of Computer Science and Technology, East China Normal University
{lip21,qtang22}@m.fudan.edu.cn, ybwu@cs.ecnu.edu.cn
{txsun19,hyan19,xjhuang,xpqiu}@fudan.edu.cn

Abstract

Large language models (LLMs) pre-trained on massive corpora have demonstrated impressive few-shot learning ability on many NLP tasks. A common practice is to recast the task into a text-to-text format such that generative LLMs of natural language (NL-LLMs) like GPT-3 can be prompted to solve it. However, it is non-trivial to perform information extraction (IE) tasks with NL-LLMs since the output of the IE task is usually structured and therefore is hard to be converted into plain text. In this paper, we propose to recast the structured output in the form of code instead of natural language and utilize generative LLMs of code (Code-LLMs) such as Codex to perform IE tasks, in particular, named entity recognition and relation extraction. In contrast to NL-LLMs, we show that Code-LLMs can be well-aligned with these IE tasks by designing code-style prompts and formulating these IE tasks as code generation tasks. Experiment results on seven benchmarks show that our method consistently outperforms fine-tuning moderate-size pre-trained models specially designed for IE tasks (e.g., UIE) and prompting NL-LLMs under few-shot settings. We further conduct a series of in-depth analyses to demonstrate the merits of leveraging Code-LLMs for IE tasks.[1]

1 Introduction

Information extraction (IE) aims to recognize structured information from plain text. It spans various tasks with diverse output structures such as named entity recognition (NER), relation extraction (RE), etc. (Sang and Meulder, 2003; Grishman, 2019; Wang et al., 2021a; Zhong and Chen, 2021; Lu et al., 2022). To express and address these different tasks in a unified framework, recent works propose to linearize the output structures into unstructured strings and solve the IE tasks with sequence generation models (Yan et al., 2021b; Huguet Cabot and Navigli, 2021; Paolini et al., 2021; Josifoski et al., 2022; Lu et al., 2022). For example, given the input sentence *"Steve became CEO of Apple in 1998 ."* of a NER task, UIE (Lu et al., 2022) generates the target as a sequence `"((person: Steve) (organization: Apple))"`.



(a) Performing NER with NL-LLMs

(b) Performing NER with Code-LLMs

Figure 1: Illustrations of performing structured NER task with NL-LLMs and Code-LLMs, respectively. In contrast to prompting NL-LLMs with plain natural language, we utilize Code-LLMs with structured code-style prompts to mitigate the output discrepancy between the pre-training and inference stages.

While this kind of linearizing approach achieves promising results with sufficient training data, it still performs poorly under the few-shot scenario. For instance, compared with full-data training, the performance dropped by around 20% when applying UIE on a 5-shot NER task CoNNL03 (Lu et al., 2022).

Considering the tremendous few-shot adapting capabilities of large language models (LLMs) (Brown et al., 2020; Rae et al., 2021;

[*] Equal contribution.

[†] Corresponding author.

[1]Code is available at https://github.com/dasepli/CodeIE

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 15339–15353
July 9-14, 2023 ©2023 Association for Computational Linguistics

Chowdhery et al., 2022; Hoffmann et al., 2022), we manage to employ them to perform few-shot IE tasks, especially the few-shot NER task and RE task. Typically, for NLP tasks like text classification, previous works reformulate them into text-to-text generation formats and prompt the LLMs of natural language (NL-LLMs) like GPT-3 (Brown et al., 2020) to generate the answer. In contrast, due to the complex structure inside the targets of IE tasks, linearized targets of previous works like `"((person: Steve) (organization: Apple))"` are usually "unnatural", resulting in a mismatch between the output format at the pre-training time and the inference time (see Figure 1(a)). As a consequence, when using these flattening methods to perform IE tasks with pre-trained language models, the predicted outputs are fragile and often require complex decoding strategies to be post-processed into valid structures (Lu et al., 2022; Josifoski et al., 2022).

In this paper, we propose to frame these two IE tasks into code generation tasks and leverage the LLMs of code (Code-LLMs) to address them. We argue the abundant structured code information encoded in the pretrained Code-LLMs can benefit these IE tasks. As demonstrated in Figure 1(b), it is easy to convert the text-to-structure IE task into a structure-to-structure code generation task, while transforming it into a text-to-text format can be difficult. Take the example input in Figure 1, *"Steve became CEO of Apple in 1998 ."*, we wrap it into a piece of Python code, and formulate the structured entity outputs as Python dictionaries with keys `"text"` and `"type"`. We compose them into a Python function that is semantically equivalent to the NER example, which is shown as follows:

```python
def named_entity_recognition(input_text):
    """ extract named entities from the input_text . """
    input_text = "Steve became CEO of Apple in 1998 ."
    entity_list = []
    # extracted named entities
    entity_list.append({"text": "Steve", "type": "person"})
    entity_list.append({"text": "Apple",\
                         "type": "organization"})
```

After demonstrating a few training samples with the same format, we feed the code-style prompt (the highlighted lines with light grey color) into Code-LLMs and get the structured prediction.

We conduct experiments on seven benchmarks of NER and RE tasks, and carefully analyze the benefits of our approach (named CodeIE). The findings are as follows:

Model Type	Generative?	Extremely Large?	Structured Pre-train?	Few-Shot NER and RE Tasks
	Unified Framework	Few-shot Learning	Structured Task	
Pre. Models (e.g., UIE)	✓	✗	✓	✗
NL-LLMs (e.g., GPT-3)	✓	✓	✗	✗
Code-LLMs (e.g., Codex)	✓	✓	✓	✓

Table 1: A high-level comparison between previous IE Models, NL-LLMs and Code-LLMs. The bottom row illustrates our approach.

1) Prompting Code-LLMs (e.g., Codex (Chen et al., 2021)) with code-style inputs consistently outperforms fine-tuning UIE, a specially pre-trained model for IE tasks, and prompting NL-LLMs (e.g., GPT-3) under few-shot settings.
2) With the same LLM (either NL-LLM or Code-LLM), the code-style prompt performs better than the linearized text prompt, demonstrating the advantage of representing structured targets with code.
3) With the same prompt (either natural language or code), the Code-LLM (i.e., Codex) achieves better performance than the NL-LLM (i.e., GPT-3), demonstrating the merits of performing IE tasks with Code-LLMs.
4) Compared with natural language prompts, using the code-style prompts showed higher fidelity to the output structures, i.e., the outputs have a lower structural error rate.

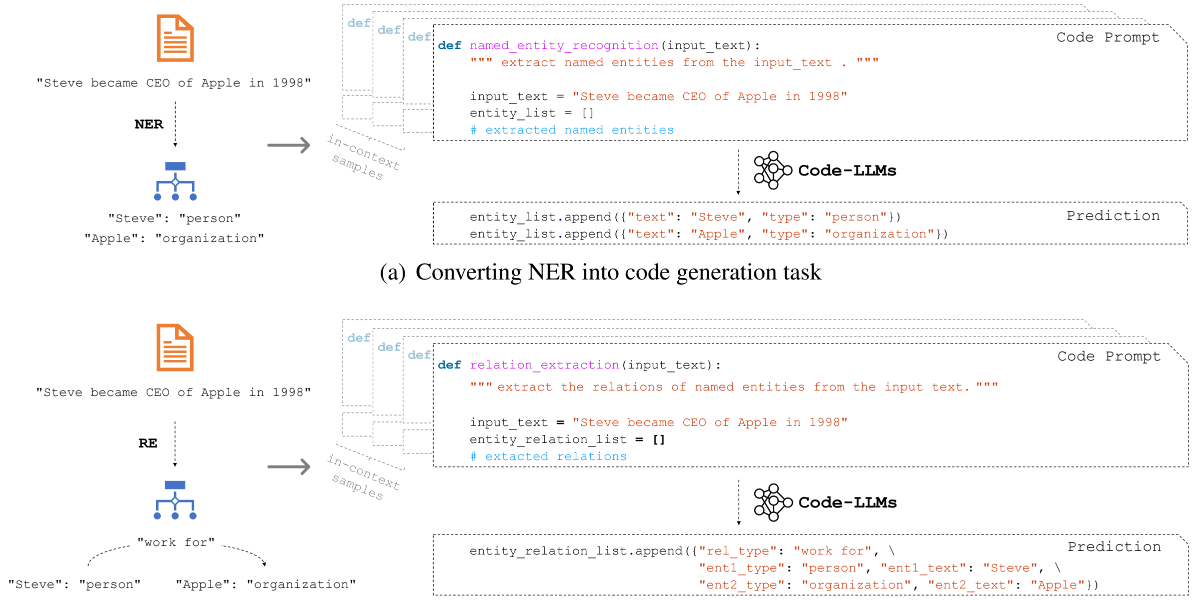
The high-level differences between previous moderate-size models, NL-LLMs, and Code-LLMs for IE tasks are summarized in Table 1.

2 CodeIE

In this section, we first formulate the two IE tasks we focus on, named entity recognition (NER) and relation extraction (RE) in Section 2.1. Then we describe how we recast these structured prediction tasks into code generation tasks (Section 2.2) and prompt Code-LLMs to perform them (Section 2.3) under the few-shot scenario. We use Python language for our code generation tasks since public Python codebases are abundant and Code-LLMs are sufficiently pre-trained on them.

2.1 Task Formulation

Given an input sentence x with l tokens $x_1, x_2, \ldots, x_l$, IE tasks are to predict structured target y from x.



```
def named_entity_recognition(input_text):
    """ extract named entities from the input_text . """

    input_text = "Steve became CEO of Apple in 1998"
    entity_list = []
    # extracted named entities
```

```
    entity_list.append({"text": "Steve", "type": "person"})
    entity_list.append({"text": "Apple", "type": "organization"})
```

(a) Converting NER into code generation task

```
def relation_extraction(input_text):
    """ extract the relations of named entities from the input text. """

    input_text = "Steve became CEO of Apple in 1998"
    entity_relation_list = []
    # extacted relations
```

```
    entity_relation_list.append({"rel_type": "work for", \
                                 "ent1_type": "person", "ent1_text": "Steve", \
                                 "ent2_type": "organization", "ent2_text": "Apple"})
```

(b) Converting RE into code generation task

Figure 2: The way to convert NER and RE into code generation task.

The target y of NER is a set of (e, t) pairs, where e is an entity span (e.g., "Steve") and t is the corresponding entity type (e.g., "person"). The entity span is a sequence of tokens from x, and the entity type belongs to a pre-defined entity type set T.

The prediction target y of RE is comprised of a set of triplets (e_1, r, e_2), where e_1 and e_2 are two entity spans from x and $r \in R$ is the semantic relation (e.g., "work for") between the two entities. Here R denotes a pre-defined relation type set. In addition to extracting the relation of entities, we are often interested in also predicting the entity types t_1 and t_2 of entities e_1 and e_2 at the same time.

In the few-shot setting, we are given a small set of annotated samples $\{(x_i, y_i)\}_{i=1}^{n}$ that consists of k samples per class to compose a k-shot setting.

2.2 Formulating IE Tasks into Code Generation Task

In order to utilize generative Code-LLMs for IE tasks, we reformulate IE tasks as code generation tasks. The code generation task is to predict the subsequent code sequence given an incomplete piece of code. Hence, we can recast the input and output of the IE task into an incomplete piece of code and the code to be predicted, respectively, such that they can compose a complete piece of code that is semantically equivalent to the original sample while maintaining the syntax of the programming language.

In this work, we mainly use Python functions to represent IE tasks. We wrap the input text x into a code-style prompt x^c and represent the output structure y with structured Python elements, such as the list, dictionary, etc. As shown in Figure 2, for NER and RE tasks, we first transform the task name into the name of the Python function and add a docstring to illustrate the goal of the task. We assign the input text string x to a variable input_text. Then we initialize an empty list to save the output and append a descriptive comment like "# extracted named entities" to prompt Code-LLMs to put named entities into the list. We pack the above code as our code prompt x^c.

For the structured target y, we utilize the append method of Python list and represent each basic information unit (e.g., a pair for NER tasks or a triplet for RE tasks) as a Python dictionary. Hence, the target y^c to be predicted by Code-LLMs is reformulated into a list of dictionaries. For NER, we add Python dictionaries with keys "text" and "type" to the list, where the values of the dictionaries are the corresponding entity span and entity type. For RE, we similarly add dictionaries with keys "rel_type", "ent1_type", "ent1_text", "ent2_type", and "ent2_text" to the list to represent the structured target.

The Code-LLM is expected to complete the list conditioning on the function name, docstring, and input text. Figure 2 shows examples of formulating an original IE sample into a code-style one.

It is worth noting that the design space of the code-style prompt is large and hard to be fully explored. The formulation described above is a straightforward instance using Python. We also explore several other formulations to recast IE tasks into code generation tasks, which can be found in Appendix A.1.

2.3 Prompting Code-LLMs with In-Context Demonstrations

Despite the carefully designed prompt, it is nontrivial to perform IE tasks by prompting Code-LLMs without any samples. Therefore, it is necessary to let Code-LLMs be aware of a few labeled samples in typical few-shot settings.

With the increasing size of pre-trained language models, fine-tuning is becoming more and more expensive or even infeasible since recent LLMs are usually released as black-box APIs (Sun et al., 2022). Hence, instead of fine-tuning Code-LLMs on the few-shot dataset, we explore including the labeled samples in the context and performing in-context learning (Brown et al., 2020). We select n samples $\{(x_i, y_i)\}_{i=1}^{n}$ from the training dataset and convert them to corresponding code-style pairs $\{(x_i^c, y_i^c)\}_{i=1}^{n}$. We concatenate them as a string to compose the in-context demonstrations $x_1^c \oplus y_1^c \dots x_n^c \oplus y_n^c$. Given an arrived test sample x, we first convert it to a code prompt x^c and prepend the demonstration context, i.e., $x_1^c \oplus y_1^c \dots x_n^c \oplus y_n^c \oplus x^c$. After feeding the constructed input into the Code-LLM, we are expected to get an output y^c that is formatted as the same as y_1^c, y_2^c, $\dots y_n^c$ (see Figure 2). We find that y^c almost always retains the syntax of Python language and is easy to be converted back to its original structure y.

3 Experiments

3.1 Setup

Datasets We evaluate our approach on NER task with CoNLL03 (Sang and Meulder, 2003), ACE04 (Doddington et al., 2004) and ACE05-E(Walker et al., 2006). For relation extraction, we evaluate on datasets CoNLL04 (Roth and Yih, 2004), ACE05-R (Walker et al., 2006), NYT (Riedel et al., 2010) and SciERC (Luan et al., 2018). Table 2 shows the dataset statistics. We follow Lu et al. (2022) to preprocess all these datasets.

	|Ents|	|Rels|	#Train	#Val	#Test
CoNLL03	4	-	14,041	3,250	3,453
ACE04	7	-	6,202	745	812
ACE05-E	7	-	7299	971	1060
CoNLL04	4	5	922	231	288
ACE05-R	7	6	10,051	2,420	2,050
NYT	3	24	56,196	5,000	5,000
SciERC	6	7	1,861	275	551

Table 2: Statistics of the datasets used in our experiments. |Ents| and |Rels| denote the number of entity types and relation types. #Train, #Val and #Test denote the sample number in each split.

Code-LLMs For Code-LLMs, we conduct experiments mainly with the `code-davinci-002` version Codex from OpenAI. Codex is a large language model adapted from GPT-3 and further pre-trained on open-source codebases. The `code-davinci-002` version Codex supports 8k input tokens at most. We get the model predictions by querying OpenAI API[2] in the few-shot in-context prompting way. We generate up to 280 tokens with greedy decoding.

Baselines We compare our approach with two kinds of few-shot learning methods:

1) **Fine-tuning** We fine-tune the base and large versions of two moderate-size pre-trained models: T5 and UIE. T5 is a sequence-to-sequence model pre-trained on large-scale text corpora. UIE is a model further pre-trained from T5 on the structured datasets. UIE utilizes the textual structured extraction language (SEL) to express the output structures. We use the same approach and parameters with Lu et al. (2022) when fine-tuning T5 and UIE.
2) **Prompting** We compare our approach with prompting NL-LLMs, in particular GPT-3. We mainly experiment with the `text-davinci-002`. We use a text prompt, of which the format is slightly modified from SEL. As shown in Figure 1(a), given an input text x, the text prompt and output format are like `"The text is` x`. The named entities in the text: "` and `"((person: ...)(organization:...))"`, respectively. See Appendix A.2 for more details of the text prompt. The approach and super-parameters of NL-LLMs prompting and Code-LLMs prompting are identical.

[2] https://openai.com/api

Model	Prompt Type	Entity			Relation				AVG
		CoNLL03	ACE04	ACE05-E	CoNLL04	ACE05-R	NYT	SciERC	
Full Data									
Pre. SoTA	-	**93.21**	86.84	84.74	73.60	65.60	92.70	35.60	76.04
UIE-large	text	92.99	**86.89**	**85.78**	**75.00**	**66.06**	-	**36.53**	-
Few Shot									
#shot (#sample)		5 (25)	2 (16)	2 (16)	5 (25)	2 (14)	1 (24)	2 (16)	
T5-base	text	33.68$_{\pm 29.17}$	7.25$_{\pm 12.00}$	9.09$_{\pm 15.74}$	14.56$_{\pm 13.87}$	0.00$_{\pm 0.00}$	5.59$_{\pm 9.68}$	0.00$_{\pm 0.00}$	10.02
UIE-base	text	70.37$_{\pm 0.54}$	44.31$_{\pm 1.61}$	39.71$_{\pm 0.91}$	45.63$_{\pm 1.50}$	8.69$_{\pm 1.41}$	-	5.69$_{\pm 0.49}$	-
T5-large	text	53.08$_{\pm 7.71}$	24.67$_{\pm 5.26}$	24.31$_{\pm 4.74}$	10.03$_{\pm 8.75}$	1.41$_{\pm 0.74}$	15.29$_{\pm 8.76}$	0.25$_{\pm 0.43}$	18.43
UIE-large	text	**70.62**$_{\pm 3.22}$	**45.08**$_{\pm 3.63}$	**43.03**$_{\pm 2.26}$	**47.68**$_{\pm 2.29}$	**9.59**$_{\pm 4.89}$	-	**7.30**$_{\pm 2.01}$	-
GPT-3	text	68.84$_{\pm 1.29}$	45.51$_{\pm 0.23}$	48.93$_{\pm 0.49}$	39.67$_{\pm 2.44}$	5.13$_{\pm 1.24}$	16.07$_{\pm 4.67}$	4.39$_{\pm 0.98}$	32.65
GPT-3	code	81.00$_{\pm 1.49}$	53.44$_{\pm 1.44}$	52.98$_{\pm 1.53}$	51.33$_{\pm 1.34}$	12.33$_{\pm 2.06}$	24.81$_{\pm 1.90}$	4.67$_{\pm 0.67}$	40.08
Codex	text	72.66$_{\pm 0.66}$	49.58$_{\pm 1.37}$	49.55$_{\pm 1.14}$	47.30$_{\pm 2.25}$	10.08$_{\pm 2.06}$	24.63$_{\pm 6.74}$	5.40$_{\pm 2.65}$	37.03
Codex	code	**82.32**$_{\pm 0.37}$	**55.29**$_{\pm 0.37}$	**54.82**$_{\pm 2.09}$	**53.10**$_{\pm 2.02}$	**14.02**$_{\pm 3.27}$	**32.17**$_{\pm 1.46}$	**7.74**$_{\pm 1.54}$	**42.78**

Table 3: Experiment performances on NER and RE benchmarks. Our approach is highlighted with light grey. The full data fine-tuning performances come from UIE. For the few-shot setting, we evaluate T5-base, UIE-base, T5-large and UIE-large with fine-tuning, and GPT-3 and Codex by few-shot prompting with two different prompt types. The text prompt is the structured extraction language (SEL) introduced by UIE. The code format is introduced in Section 2.2. We set the shot number (#shot) and the corresponding sample number (#sample) differently to fit into the GPT-3 maximum length limitation (4097 tokens).

Few-Shot Setting For each IE task, we randomly sample k training samples for each entity or relation type to construct a k-shot training set. The value of k varies across different datasets to satisfy the maximum length limitation (4097) of GPT-3. To be compatible with datasets that contain samples with empty targets, we regard those empty-target samples as an additional class and include k samples belonging to that class in the training set.

Evaluation Same as previous work (Lu et al., 2022), we use **Entity F1** and **Relation Strict F1** as the evaluation metrics for NER tasks and RE tasks, respectively. Under these metrics, an entity span prediction is correct if its offsets and entity type match the golden entity. And a relation prediction is correct if the relation type is correct and the corresponding offsets and types of its entities are correct. Since few-shot training is of high variance, we perform 3 runs with different random seeds for each experiment and report the mean and standard deviation of the metric.

3.2 Results

LLMs vs. Moderate-sized Models As shown in Table 3, LLMs (GPT-3 and Codex) achieve superior performance over moderate-sized models (T5 and UIE) under few-shot settings, demonstrating a strong few-show learning ability on IE tasks. Especially, on average performance over the seven considered benchmarks, our proposed CodeIE (Codex + code prompt) achieves the best results, improving T5-large and T5-base by 132% and 327%, respectively. In addition, under 1-shot learning settings, CodeIE improves the performance of UIE-large by more than 60% on CoNLL03 and CoNLL04 benchmarks (see Table 6 in the Appendix).

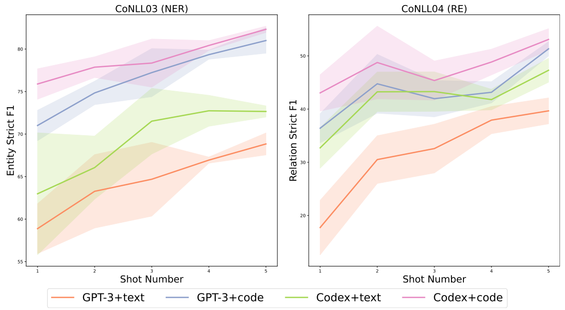


Figure 3: Performance with different shot numbers on CoNLL03 (NER) and CoNLL04 (RE) datasets.

Code Prompt vs. Text Prompt We then compare the performance of code prompt vs. text prompt when using the same LLM, i.e., comparing ⟨GPT-3 + text prompt⟩ with ⟨GPT-3 + code prompt⟩ and comparing ⟨Codex + text prompt] with ⟨Codex + code prompt⟩. As a result, we find that prompting LLMs with code yields significant improvement (23% for GPT-3 and 16% for Codex). What is surprising is that code prompt is even more beneficial to GPT-3, which is not specif-

Model	Prompt Design	Entity CoNLL03	Relation CoNLL04
GPT-3	struct lang	68.84$_{\pm 1.29}$	39.67$_{\pm 2.44}$
	natural lang	46.36$_{\pm 12.56}$	40.90$_{\pm 3.67}$
Codex	func def	82.32$_{\pm 0.37}$	53.10$_{\pm 2.02}$
	class init	81.29$_{\pm 0.72}$	52.32$_{\pm 0.94}$
	func exec	84.05$_{\pm 1.24}$	53.32$_{\pm 3.47}$
	func init-	81.95$_{\pm 1.01}$	53.59$_{\pm 1.10}$

Table 4: Performance of different prompt designs. "struct lang" and "func def" are the "text" and "code" prompt types respectively in our main experiments.

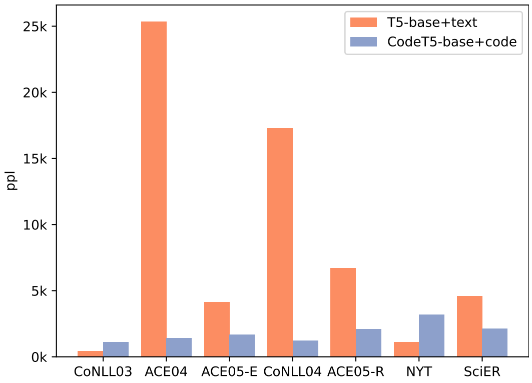


Figure 4: Format consistency between the input format and the model (measured by perplexity) for text prompt and code prompt on 7 datasets.

ically trained on code data.

Code-LLMs vs. NL-LLMs When using the same kind of prompt and comparing the used LLMs, i.e., comparing ⟨GPT-3 + text prompt⟩ and ⟨Codex + text prompt⟩ and comparing ⟨GPT-3 + code prompt⟩ and ⟨Codex + code prompt⟩, we find that Codex consistently surpasses GPT-3, demonstrating that code pre-training can be beneficial to IE tasks.

Different Shot Numbers We further compare these approaches under different shot numbers on CoNLL03 and CoNLL04. As shown in Figure 3, we can see that the obtained phenomenons still hold when increasing the number of shots.

Different Prompt Designs The design of the prompt can be an important factor affecting the model performance (Min et al., 2022). Hence, we explore additional prompt designs for both text prompt and code prompt. The detailed prompt deigns can be found in Appendix A. The experimental results are shown in Table 4, from which we find that code prompts consistently outperform text prompts. Hence, the superior performance of using code prompts is mainly contributed by the code style instead of some specific instance of prompt design.

Different LLMs To verify the versatility of the proposed approach and the observed findings, we further conduct experiments with `text-davinci-001` version of GPT-3 and `code-davinci-001` version of Codex. As shown in Table 7, the previous findings still hold across the two different versions.

4 Analysis

To take a closer look at the difference between prompting NL-LLMs with textual format input and prompting Code-LLMs with code format input, in this section, we define several informative metrics and conduct in-depth analyses to shed some light on the following question: *what contributes to the final performance of* CODEIE *for IE tasks?*

4.1 Format Consistency

We can see from Figure 1(a) that an apparent inappropriateness to use NL-LLMs for IE tasks is the inconsistency between the structured output format at inference time and NL-LLMs that are trained on natural language at pre-training time, while the format of code-style output aligns well with Code-LLMs. It has been evidenced that adapting pre-trained models to downstream tasks in a manner that is well aligned with it pre-training paradigm usually achieves better few-shot learning performance. Hence we assume *the promising performance of* CODEIE *partly comes from the better format consistency between the code-style sample and the pretrained code model.*

To verify this hypothesis, given a sample, we compare the perplexities of a pre-trained language model on its text format and a pre-trained code model on its code format. Formally, given a generative model M, the conditional perplexity *ppl* of a sample (x, y) is as follows,

$$ppl_M(y|x) = \prod_{i=1}^{m} P_M(y_i|y_1 \cdots y_{i-1}, x)^{-\frac{1}{l}}. \tag{1}$$

For an original IE sample (x, y), we first transform it to its natural language text pair (x^t, y^t) and its code piece pair (x^c, y^c), and then compute the conditional perplexity of them with the language model M^{nl} and the code model M^c, respectively, i.e., the

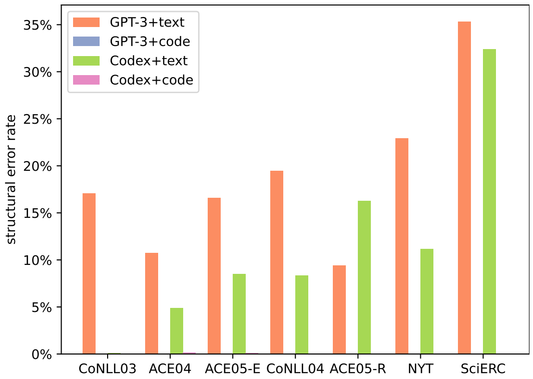


Figure 5: Structural error rate of different combinations of LLM and prompting methods. Prompting LLMs with code exhibits higher structure fidelity.

$ppl_{M^{nl}}(y^t|x^t)$ and $ppl_{M^c}(y^c|x^c)$. A lower conditional perplexity means the output format aligns well with the pre-training distribution of the model.

Since LLMs usually limit user access by their black-box APIs, we instead utilize two agent models T5 (Raffel et al., 2020) and CodeT5 (Wang et al., 2021b) to calculate the perplexities. CodeT5 is a variant of T5 model that is further pre-trained on code data. We calculate the perplexities on the previous seven datasets with the base verison of the two models, namely T5-base and CodeT5-base. Figure 4 shows the mean perplexities of two base version models on the training samples of each task. We can observe the perplexity of the text format outputs measured by T5-base is usually larger than code format outputs measured by CodeT5-base. That means, transforming IE samples to code formats can better align with the data distribution of code pre-training and therefore the pre-trained code language model.

4.2 Model Fidelity

Besides the low format consistency of prompting ML-LLMs, we find that NL-LLMs are more likely to generate outputs with structural and semantic errors when performing few-shot IE tasks than Code-LLMs. In other words, *Code-LLMs seem to be more faithful to the demonstrated few-shot samples than NL-LLMs*. To quantitatively measure the model fidelity, we define two metrics:

Structure Fidelity Structure fidelity measures how faithful the model is to the structure of demonstrations provided in the context. This can be simply measured by calculating the structural error rate, which is the proportion of generated samples with structural errors. In particular, we construct a parser with a series of hand-written rules to transform the model-generated outputs back to the desired format and filter out samples with invalid structures. Figure 5 demonstrates the structure fidelity of different models with different prompts on the seven benchmarks. Results show that the outputs generated by GPT-3 and Codex using text prompts are fragile while using code prompts tends to generate nearly zero structurally erroneous samples. Besides, with the same text prompt, Codex tends to generate fewer structurally errant samples than GPT-3, demonstrating its superior understanding ability on general structured input instead of being limited to existing programming languages.

Task	Error Type	Samples
NER	Entity type not in T	currency, company, time, event, profession,organizational indicator, finanical, object, event
RE	Relation type not in R	called, organization, person, relate, specialize, assumption, cause, assign

Table 5: Semantically errant samples detected in our experiments. These errant samples mainly came from GPT-3.

Semantic Fidelity Another measurement of model fidelity is semantic fidelity, which is designed for those samples that have a valid structure and can succeed in our parser but are semantically incorrect. The difference between the defined semantic fidelity and the conventional prediction error is that semantic fidelity mainly considers model behaviours that violate the formulation of the task, e.g., predicting an entity type that does not exist in the given entity type set or extracting an entity span that does not appear in the input text. Some example semantic errors detected in our experiments are listed in Table 5. We report the statistical result of the tasks in Table 8 and Table 9 in the Appendix. As a result, we find that GPT-3 generated more semantic errors than Codex though some of the errors seem to be "correct" but are out of the pre-defined class set. In a nutshell, GPT-3 tends to generate free-form results and Codex is more faithful to the demonstrations provided in the context and therefore is more predictable for IE tasks.

4.3 Fine-grained Performance

In addition, we conduct a fine-grained evaluation to compare different approaches. In addition to the F1 score, precision and recall are also important metrics for NER and RE tasks. To investigate

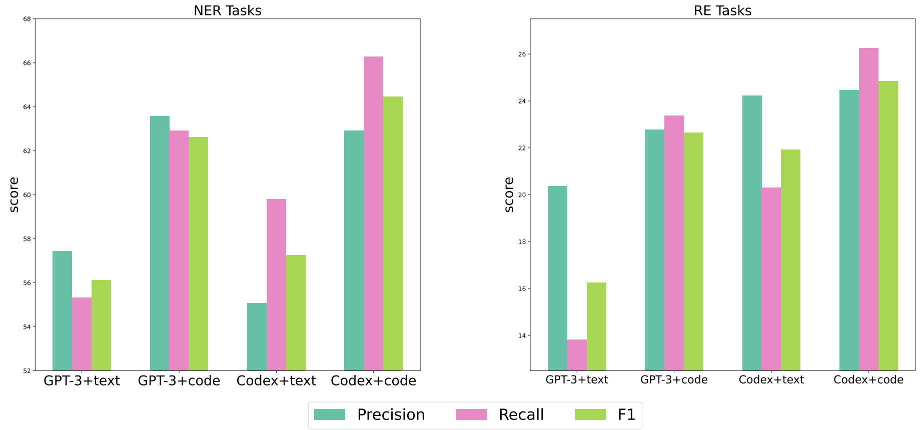


Figure 6: Model Performance Details on NER and RE Tasks. We report the averaged metric scores of all the NER or RE datasets.

how different LLMs and prompting methods affect precision and recall, we report the two metrics in Figure 6. Results show that: (a) The code prompt improves model performance in both precision and recall; (b) Compared with GPT-3, Codex achieves higher recall and comparable precision on NER tasks and and achieves both higher precision and recall on RE tasks.

5 Related Work

Generative Information Extraction Generative information extraction which frames IE tasks as token generation tasks receive more attention recently due to their potential to unify different tasks (Yan et al., 2021a; Josifoski et al., 2022). Yan et al. (2021a) designs various ways to linearize entities into a sentence to unify various named entity recognition subtasks. TANL (Paolini et al., 2021) uses augmented language to improve the effect of generative models. Lu et al. (2022) also proposes a structured extraction language (SEL) and pre-trains their UIE model with this language on multiple structured datasets. These works linearize the structure output of IE tasks into text format to align the pre-trained models. Different from them, we propose to recast the structural samples of IE tasks into structural code format and utilize aligned pre-trained code models to perform the tasks.

Code-LLMs for Complex Tasks Recent works show Code-LLMs perform better on complex tasks like commonsense and symbolic reasoning (Madaan et al., 2022; Cheng et al., 2022), mathematical logic (Suzgun et al., 2022) and event argument prediction (Wang et al., 2022) tasks. We focus on the two mainstream IE tasks different from them, i.e., NER and RE. Besides, in-depth analyses are conducted to provide more insights.

LLMs for Few-Shot NER and RE While LLMs like GPT-3 have shown strong few-shot learning abilities in many NLP tasks, limited works have explored their capabilities on typical IE tasks like NER and RE. Epure and Hennequin (2021) evaluate GPT-2 (Radford et al., 2019) on open-domain NER tasks with few-shot demonstrating. A recent work (Gutiérrez et al., 2022) tests the performance of GPT-3 on biomedical NER and RE tasks and finds it underperforms compared to fine-tuning smaller pretrained models. Its concurrent work (Agrawal et al., 2022) finds that GPT-3 performs well on few-shot clinical IE tasks. We conduct our experiments on more general NER and RE datasets and find GPT-3 can achieve comparable performance to fine-tuning the UIE model. Besides, we successfully employ the LLMs of code with better performances for these IE tasks.

6 Conclusion

We propose the first work to utilize the structured Code-LLMs with code-style prompts to perform the few-shot NER and RE tasks. Experiments show our approach consistently surpasses the UIE models and the NL-LLMs counterpart under the few-shot setting. We conducted extensive analysis and find the performances come from better format consistency and model fidelity, etc. We think these analyzes can facilitate future work. As the further works, we will employ CODEIE on more IE tasks in different domains, and inspect the robustness of it.

Limitations

Though our approach demonstrates better performances than the baseline models, how to design a good code-format prompt has not been fully inspected. Besides, we mainly conduct experiments on the black-box GPT-3 and Codex models but they are not open-sourced and querying the GPT-3 model cost the economic budget. And the use of LLMs may bring environmental pollution. Another limitation of our approach is that the Code-LLMs mainly trained on programming language datasets with English annotations. Exploring our model on non-English datasets (like Chinese datasets) is the future work.

Acknowledgements

We would like to express our gratitude to the reviewers for their helpful comments and suggestions. We are also very grateful to Yaojie Lu for his friendly assistance during our experiments. This work was supported by the National Natural Science Foundation of China (No. 62236004 and No. 62022027) and CCF-Baidu Open Fund.

References

Monica Agrawal, Stefan Hegselmann, Hunter Lang, Yoon Kim, and David A. Sontag. 2022. Large language models are few-shot clinical information extractors. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 1998–2022. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2022. Binding language models in symbolic languages. *CoRR*, abs/2210.02875.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

George R. Doddington, Alexis Mitchell, Mark A. Przybocki, Lance A. Ramshaw, Stephanie M. Strassel, and Ralph M. Weischedel. 2004. The automatic content extraction (ACE) program - tasks, data, and evaluation. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*. European Language Resources Association.

Elena V. Epure and Romain Hennequin. 2021. A realistic study of auto-regressive language models for named entity typing and recognition. *CoRR*, abs/2108.11857.

Ralph Grishman. 2019. Twenty-five years of information extraction. *Nat. Lang. Eng.*, 25(6):677–692.

Bernal Jiménez Gutiérrez, Nikolas McNeal, Clay Washington, You Chen, Lang Li, Huan Sun, and Yu Su. 2022. Thinking about GPT-3 in-context learning for biomedical ie? think again. *CoRR*, abs/2203.08410.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models. *CoRR*, abs/2203.15556.

Pere-Lluís Huguet Cabot and Roberto Navigli. 2021. REBEL: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Martin Josifoski, Nicola De Cao, Maxime Peyrard, Fabio Petroni, and Robert West. 2022. GenIE: Generative information extraction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4626–4643, Seattle, United States. Association for Computational Linguistics.

Yaojie Lu, Qing Liu, Dai Dai, Xinyan Xiao, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2022. Unified structure generation for universal information extraction. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5755–5772, Dublin, Ireland. Association for Computational Linguistics.

Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. 2018. Multi-task identification of entities, relations, and coreference for scientific knowledge graph construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3219–3232, Brussels, Belgium. Association for Computational Linguistics.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. *CoRR*, abs/2210.07128.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? *CoRR*, abs/2202.12837.

Giovanni Paolini, Ben Athiwaratkun, Jason Krone, Jie Ma, Alessandro Achille, Rishita Anubhai, Cícero Nogueira dos Santos, Bing Xiang, and Stefano Soatto. 2021. Structured prediction as translation between augmented natural languages. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew J. Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *CoRR*, abs/2112.11446.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2010, Barcelona, Spain, September 20-24, 2010, Proceedings, Part III*, volume 6323 of *Lecture Notes in Computer Science*, pages 148–163. Springer.

Dan Roth and Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning, CoNLL 2004, Held in cooperation with HLT-NAACL 2004, Boston, Massachusetts, USA, May 6-7, 2004*, pages 1–8. ACL.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003*, pages 142–147. ACL.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 20841–20855. PMLR.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *CoRR*, abs/2210.09261.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. Ace 2005 multilingual training corpus ldc2006t06. In *Philadelphia: Linguistic Data Consortium*. Web Download.

Xingyao Wang, Sha Li, and Heng Ji. 2022. Code4struct: Code generation for few-shot structured prediction from natural language. *CoRR*, abs/2210.12810.

Yijun Wang, Changzhi Sun, Yuanbin Wu, Hao Zhou, Lei Li, and Junchi Yan. 2021a. Unire: A unified label space for entity relation extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 220–231. Association for Computational Linguistics.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021b. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8696–8708, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hang Yan, Junqi Dai, Tuo Ji, Xipeng Qiu, and Zheng Zhang. 2021a. A unified generative framework for aspect-based sentiment analysis. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2416–2429, Online. Association for Computational Linguistics.

Hang Yan, Tao Gui, Junqi Dai, Qipeng Guo, Zheng Zhang, and Xipeng Qiu. 2021b. A unified generative framework for various NER subtasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5808–5822, Online. Association for Computational Linguistics.

Zexuan Zhong and Danqi Chen. 2021. A frustratingly easy approach for entity and relation extraction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 50–61. Association for Computational Linguistics.

A Prompt Format Design

A.1 Code Format Prompts

We design several code-style prompt formats. We use the input sentence *"Steve became CEO of Apple in 1998 ."* and the corresponding entities ("Steve": person, "Apple": organization) and relations ("work for" of the two entities "Steve" and "Apple") as a running sample for the NER and RE tasks.

The name of the format design is denoted with different `font`. we demonstrate the Python format prompt for NER and RE tasks. The prompt part is highlighted with grey color and the following codes are the expected output. We list the designed format as follows:

`1. func def:` our main code format prompt to transform the IE tasks into code formats.

For the NER task, the format is

```
def named_entity_recognition(input_text):
    """ extract named entities from the input_text . """
    input_text = "Steve became CEO of Apple in 1998 ."
    entity_list = []
    # extracted named entities
    entity_list.append({"text": "Steve", "type":
    ↪  "person"})
    entity_list.append({"text": "Apple","type":
    ↪  "organization"})
```

For the RE task, the format is

```
def relation_extraction(input_text):
    """ extract the relations of named entities from the
    ↪  input_text . """
    input_text = "Steve became CEO of Apple in 1998"
    entity_relation_list = []
    # extacted relations
    entity_relation_list.append({"rel_type": "work for",
    ↪  "ent1_type": "person", "ent1_text": "Steve",
    ↪  "ent2_type": "organization", "ent2_text":
    ↪  "Apple"})
```

`2. class init:` we describe the IE tasks with the Python class.

For the NER task, the format is

```
class NamedEntityRecognition:
    """ extract named entities from the input_text . """
    def __init__(self, input_text):
        self.input_text = "Steve became CEO of Apple in
        ↪  1998 ."
        entity_list = []
        # extracted named entities
        entity_list.append({"text": "Steve", "type":
        ↪  "person"})
        entity_list.append({"text": "Apple",\
                            "type": "organization"})
```

For the RE task, the format is

```
class RelationExtraction:
    """ extract the relations of named entities from the
    ↪  input_text . """
    def __init__(self, input_text):
        self.input_text = "Steve became CEO of Apple in
        ↪  1998"
        entity_relation_list = []
        # extacted relations
        entity_relation_list.append({"rel_type": "work
        ↪  for", "ent1_type": "person", "ent1_text":
        ↪  "Steve", "ent2_type": "organization",
        ↪  "ent2_text": "Apple"})
```

3. `func exec`: describe the IE tasks as a function execution procedure.

For the NER task, the format is

```
# extract named entities from a sentence .
input_text = "Steve became CEO of Apple in 1998 ."
output = named_entity_recognition(input_text)
# the output is
# {"text": "Steve", "type": "person"}
# {"text": "Apple", "type": "organization"}
```

For the RE task, the format is

```
# extract the relations of named entities from from a
↪  sentence .
input_text = "Steve became CEO of Apple in 1998"
output = relation_extraction(input_text)
# the output is
# {"rel_type": "work for", "ent1_type": "person",
↪  "ent1_text": "Steve", "ent2_type": "organization",
↪  "ent2_text": "Apple"}
```

4. `func init-`: perturb the rational format design by exchanging the format design of NER and RE tasks.

For the NER task, the format is

```
def relation_extraction(input_text):
    """ extract the relations of named entities from the
    ↪  input_text . """
    input_text = "Steve became CEO of Apple in 1998 ."
    entity_relation_list = []
    # extracted relations
    entity_relation_list.append({"text": "Steve", "type":
    ↪  "person"})
    entity_relation_list.append({"text": "Apple","type":
    ↪  "organization"})
```

For the RE task, the format is

```
def named_entity_recognition(input_text):
    """ extract named entities from the input_text . """
    input_text = "Steve became CEO of Apple in 1998"
    entity_list = []
    # extracted named entities
    entity_list.append({"rel_type": "work for",
    ↪  "ent1_type": "person", "ent1_text": "Steve",
    ↪  "ent2_type": "organization", "ent2_text":
    ↪  "Apple"})
```

A.2 Text Format Prompts

Similar to the above section (A.1), we describe the textual format prompt we used given the text input *"Steve became CEO of Apple in 1998 ."*. The text input prompts are all the same and we highlighted the expected outputs with blue colour.

1. `struct lang`: our mainly used text format prompt.

For the NER task, the transformed format is:

The text is "Steve became CEO of Apple in 1998 .". The named entities in the text: `((person: Steve)(organization: Apple))`

For the RE task, the transformed format is:

The text is "Steve became CEO of Apple in 1998 .". The relations of named entities in the text: `((person: Steve (work for: Apple)) (organization: Apple))`

2. `natural lang`: a more "natural" format to describe the structures in natural language.

For the NER task, the transformed format is:

The text is "Steve became CEO of Apple in 1998 .". The named entities in the text: `"Steve" is "person". "Apple" is "organization".`

For the RE task, the transformed format is:

The text is "Steve became CEO of Apple in 1998 .". The relations of named entities in the text: `person "Steve" work for organization "Apple".`

Model	**Prompt Type**	**Entity**			**Relation**			
		CoNLL03	**ACE04**	**ACE05-E**	**CoNLL04**	**ACE05-R**	**NYT**	**SciERC**
UIE-large	text	46.75$_{\pm 6.13}$	35.25$_{\pm 2.31}$	34.29$_{\pm 1.93}$	25.81$_{\pm 5.93}$	5.15$_{\pm 4.43}$	-	4.65$_{\pm 0.61}$
Codex	code	**75.89**$_{\pm 1.79}$	**54.27**$_{\pm 2.14}$	**51.91**$_{\pm 2.51}$	**43.05**$_{\pm 3.42}$	**7.09**$_{\pm 4.40}$	**32.17**$_{\pm 1.46}$	**6.05**$_{\pm 0.82}$
RoI↑		62.33 %	53.96 %	51.39 %	66.80 %	37.67 %	-	30.11 %

Table 6: The 1-shot results of UIE-large and Codex, and Rate of Increase (RoI) of Codex than UIE-large.

Model	**Prompt Type**	**Entity**			**Relation**			
		CoNLL03	**ACE04**	**ACE05-E**	**CoNLL04**	**ACE05-R**	**NYT**	**SciERC**
`text-davinci-002`	text	68.84$_{\pm 1.29}$	45.51$_{\pm 0.23}$	48.93$_{\pm 0.49}$	39.67$_{\pm 2.44}$	5.13$_{\pm 1.24}$	16.07$_{\pm 4.67}$	4.39$_{\pm 0.98}$
`code-davinci-002`	code	**82.32**$_{\pm 0.37}$	**55.29**$_{\pm 0.37}$	**54.82**$_{\pm 2.09}$	**53.10**$_{\pm 2.02}$	**14.02**$_{\pm 3.27}$	**32.17**$_{\pm 1.46}$	**7.74**$_{\pm 1.54}$
`text-davinci-001`	text	38.55$_{\pm 6.11}$	29.23$_{\pm 1.49}$	29.73$_{\pm 2.22}$	19.63$_{\pm 4.37}$	0.89$_{\pm 0.66}$	-	0.87$_{\pm 0.22}$
`code-davinci-001`	code	**61.86**$_{\pm 1.88}$	**33.62**$_{\pm 3.85}$	**36.26**$_{\pm 1.45}$	**28.75**$_{\pm 1.90}$	**1.65**$_{\pm 1.55}$	-	**1.91**$_{\pm 0.30}$

Table 7: Performances of different LLMs. `text-davinci-001` is an InstructGPT model based on the previous GPT-3 model with Feedback Made Easy strategy. `code-davinci-001` is an earlier version of `code-davinci-002`.

	Entity Label Error			**Entity Span Error**		
	CoNLL03	ACE04	ACE05-E	CoNLL03	ACE04	ACE05-E
#test	3453	812	1060	3453	812	1060
#in-context shot	5	2	2	5	2	2
GPT-3+text	15	298	414	113	140	114
GPT-3+code	57	755	949	28	73	57
Codex+text	3	30	64	90	88	141
Codex+code	8	536	601	18	51	37

Table 8: Detailed Errors on NER datasets. "Entity Label Error" means the predicted label is not in the predefined label set. "Entity Span Error" means the predicted span is not in the original input text. The reported error numbers are counted by summing 3 different seeds.

	Ent1 Type Error				**Ent1 Span Error**				**Relation Type Error**			
	CoNLL04	ACE05-Rel	NYT	SciERC	CoNLL04	ACE05-R	NYT	SciERC	CoNLL04	ACE05-R	NYT	SciERC
#test	288	2050	5000	551	288	2050	5000	551	288	2050	5000	551
#in-context shot	5	2	1	2	5	2	1	2	5	2	1	2
GPT-3+text	2	1078	669	335	26	266	491	160	169	617	3274	358
GPT-3+code	3	410	102	410	13	105	1029	105	6	12	2000	50
Codex+text	1	815	100	815	20	155	477	155	84	820	315	820
Codex+code	0	346	10	346	1	108	544	108	2	0	141	17

Table 9: Detailed Errors on RE datasets. "Ent1 Type Error" means the predicted entity type of the first entity is not in the predefined type set. "Ent1 Span Error" means the predicted span of the first entity is not in the original input text. "Relation Type Error" means the predicted label is not in the predefined relation type set. The reported error numbers are counted by summing 3 different seeds.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
section 'limitations"

☑ A2. Did you discuss any potential risks of your work?
section 'limitations"

☑ A3. Do the abstract and introduction summarize the paper's main claims?
section "abstract" and section 1 "introduction"

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 3

☑ B1. Did you cite the creators of artifacts you used?
Section 3

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
hard to find

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
complex procedure

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
complex procedure

☒ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
complex procedure

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
examples number, details splits

C ☑ Did you run computational experiments?

Section 3

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 3

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 3

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Section 3

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Section 3

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

□ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

□ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

□ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

□ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

□ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.