KnowCoder: Coding Structured Knowledge into LLMs for Universal Information Extraction

Zixuan Li[*†], Yutao Zeng[*], Yuxin Zuo[*], Weicheng Ren[*],
Wenxuan Liu, Miao Su, Yucan Guo, Yantao Liu, Xiang Li, Zhilei Hu, Long Bai,
Wei Li, Yidan Liu, Pan Yang, Xiaolong Jin[†], Jiafeng Guo[†], Xueqi Cheng
Key Laboratory of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences
{lizixuan, jinxiaolong, guojiafeng}@ict.ac.cn
https://ict-goknow.github.io/knowcoder.github.io/

Abstract

In this paper, we propose KnowCoder, a Large Language Model (LLM) to conduct Universal Information Extraction (UIE) via code generation. KnowCoder aims to develop a kind of unified schema representation that LLMs can easily understand and an effective learning framework that encourages LLMs to follow schemas and extract structured knowledge accurately. To achieve these, KnowCoder introduces a code-style schema representation method to uniformly transform different schemas into Python classes, with which complex schema information, such as constraints among tasks in UIE, can be captured in an LLM-friendly manner. We further construct a code-style schema library covering over **30,000** types of knowledge, which is the largest one for UIE, to the best of our knowledge. To ease the learning process of LLMs, KnowCoder contains a two-phase learning framework that enhances its schema understanding ability via code pretraining and its schema following ability via instruction tuning. After code pretraining on around 1.5B automatically constructed data, KnowCoder already attains remarkable generalization ability and achieves relative improvements by **49.8%** F1, compared to LLaMA2, under the few-shot setting. After instruction tuning, KnowCoder further exhibits strong generalization ability on unseen schemas and achieves up to **12.5%** and **21.9%**, compared to sota baselines, under the zero-shot setting and the low resource setting, respectively. Additionally, based on our unified schema representations, various human-annotated datasets can simultaneously be utilized to refine KnowCoder, which achieves significant improvements up to **7.5%** under the supervised setting.

1 Introduction

Information Extraction (IE) aims to extract explicit and structured knowledge following the manually

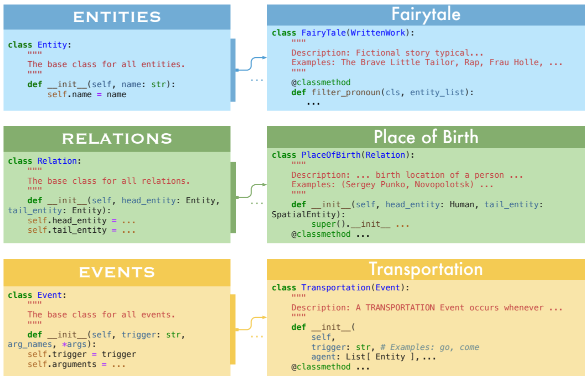

Figure 1: An illustration of KnowCoder schemas.

designed schemas. The IE schemas define high-level types of knowledge (i.e., concepts) and structures among them (Hogan et al., 2021), which include various types of entities, relations, and events. To simultaneously extract various knowledge under different schemas via a single model, the Universal Information Extraction (UIE) task is proposed (Lin et al., 2020a). Recently, Large Language Models (LLMs) have demonstrated general understanding abilities through large-scale pretraining, which drives their increasing utilization in UIE. However, their performance on UIE is still limited because of two main challenges: (1) the lack of a unified schema representation method that LLMs can easily understand; (2) the lack of an effective learning framework that encourages LLMs to accurately follow specific schemas for extracting structured knowledge.

For the first challenge, the existing UIE models first represent different schemas in a universal way, such as classification labels (Lin et al., 2020a), keywords (Gui et al., 2023), or a specifically-designed formal language (Lu et al., 2022). These schema representation methods have three main restrictions: (1) ignoring information like taxonomies (e.g., "fairytale" is a subclass of "written work") and constraints among concepts (e.g., "spouse" relation exists between two "human" entities); (2)

[*] Co-first author.
[†] Corresponding author.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 8758–8779

classification labels or a specifically designed formal language is hard for LLMs to understand and follow; (3) designed for specific IE datasets and lacking a general schema library.

To solve these restrictions, in this paper, we propose a kind of code-style schema representation method, with which various types of knowledge are generally defined as Python classes. As shown in Figure 1, the class inheritance mechanism is adopted to describe the concept taxonomies. A mechanism of type hint is employed to model constraints among different concepts. The class comments are used to provide clear definitions of concepts. And, the class methods are used to post-process the results according to specific IE guidelines. Upon this method, we construct a comprehensive code-style schema library covering over $29,000$ entity types, 900 relation types, and 500 event types based on Wikidata, the largest one for UIE, to the best of our knowledge, currently reported in the open literature.

For the second challenge, the existing learning framework for UIE directly conducts instruction tuning on LLMs to extract knowledge following specific and limited schemas (Sainz et al., 2023; Wang et al., 2023b). The enormous concepts in the constructed schema library challenge the existing training framework. To help LLMs better understand and follow these schemas, we propose an effective two-phase framework containing a schema understanding phase and a schema following phase. The former improves the ability of LLMs to understand different concepts in schemas via large-scale code pretraining on the schema definition code and corresponding instance code. The latter advances their abilities to follow specific schemas in an IE task via instruction tuning. After code pretraining on around 1.5B automatically constructed data, KnowCoder already attains remarkable generalization ability and achieves NER improvements compared to the base model, LLaMA2, by **49.8%** relative F1 point under the few-shot setting on NER. After instruction tuning on 1.5B automatically annotated data, KnowCoder experimentally demonstrates strong generalization ability on unseen schemas. Under the zero-shot setting, KnowCoder achieves average relative improvements up to **12.5%** on the NER task. Under the low-resource setting, KnowCoder gets average relative improvements up to **21.9%** on all the IE tasks. Additionally, based on our unified schema representation, various IE datasets can be simultaneously utilized to refine KnowCoder. After refinement, KnowCoder achieves consistent improvements across all IE tasks under the supervised setting, getting up to **7.5%** improvement on the relation extraction task, respectively. In general, the main contributions of this paper include:

- We propose a code-style schema representation method to uniformly represent different schemas for UIE. Using this method, we construct a large code-style schema library covering more than $30,000$ types of knowledge.
- We propose an effective learning framework for LLMs in a two-phase manner, which first enhances the schema understanding through code pretraining and then boosts schema following via instruction tuning.
- After training on billions of automatically annotated data and refining with human-annotated IE datasets, KnowCoder demonstrates superior performance on different IE tasks under the zero-shot, low-resource, and supervised settings.
- The constructed schema library, training data, code, and models are released for future research.

2 KnowCoder Schema

The proposed schema representation method uses code, a language that LLMs easy to understand, to define schemas. Specifically, KnowCoder schema adopts a series of programming language features to comprehensively model schema information, including the concept taxonomies, the constraints among different concepts, the definition of concepts, and other extraction requirements. Besides, considering that previous schema representation methods are only designed for specific datasets and contain limited types of knowledge, we further construct a large-scale schema corpus containing a wide range of knowledge.

2.1 Code-style Schema Representation Method

The code-style schema representation method comprises three basic classes, namely, "`Entity`", "`Relation`", and "`Event`". Based on the three basic classes, we represent all the concepts in the schemas by the corresponding classes. Then, the instances of each concept can be represented by the objects of the corresponding class. In the following, we will introduce four features of the proposed representation method.

Class Inheritance. We adopt the class inheritance mechanism to account for the taxonomies in the schemas. Specifically, we let class A inherit all the class members from class B if the corresponding concept A is the hyponym of concept B in the taxonomies. For a concept with multiple hypernyms, the hypernym concept with the most instances is selected. The class of an unseen concept can inherit from an existing class or directly from the basic class.

Class comment. Similar to Sainz et al. (2023), we adopt class comments to provide clear definitions of concepts. As shown in Figure 1, a class comment includes a natural language description that explains the corresponding concept and the examples of instances corresponding to that type. When there is an unseen concept, we use the description in its annotation guidelines [1] and manually give out a few examples.

Type Hint. Type hint is a formal solution to indicate the type of a value in the code. We adopt type hints in the initialization function of a class to define its constraints with other classes strictly. Thus, the constraints among the concepts in the schemas are modeled. As shown in Figure 1, taking the relation "PlaceOfBirth" for example, "`def __init__(self, head_entity: Human, tail_entity: SpatialEntity)`" denotes that the head entity must be a "`Human`" and the tail entity must be a "`SpatialEntity`".

Class Method. A class method is bound to the class and not the object of the class. They are utilized to post-process the extracted instance results of a class. For example, some IE tasks may not consider the pronouns "he" and "she" as instances of the "`Human`" concept. To address this, a class method can be added to the "`Human`" class to filter out such pronouns from the extraction results, ensuring that the output aligns with the task's unique criteria. Note that, class methods are manually designed for specific IE tasks based on their task constraints. We take a few IE datasets to demonstrate the effectiveness of class methods in our experiments, as shown in the Appendix C.

[1] If the annotation guidelines are missing, we use the description generated by GPT-4.

2.2 Schema Library Construction

We construct the code-style schema library based on Wikidata [2]. We select the concepts included in the existing IE datasets created from Wikidata, i.e., KELM (Agarwal et al., 2021), UniversalNER (Zhou et al., 2023), InstructIE (Zhang et al., 2023), and LSEE (Chen et al., 2017). We derive the constraints among concepts according to their co-occurrences. To construct the taxonomies, we extract the "`SubclassOf`" relations among these concepts from Wikidata. To obtain the description of a concept, we use its definition from Wikidata directly or generate its descriptions using GPT-4 if its definition in Wikidata is missing. Finally, the constructed schema library encompasses over $29,177$ entity types, 876 relation types, and 519 event types. The detailed statistics of the schema are in Appendix I.

3 Learning Framework of KnowCoder

To discriminate enormous concepts defined in schemas, we first let KnowCoder understand each concept through its definition and instances. Subsequently, we enhance KnowCoder to discriminate among a few concepts and extract corresponding knowledge. Thus, as shown in Figure 2, the proposed learning framework contains two phases, i.e., the schema understanding phase and the schema following phase. In the schema understanding phase, KnowCoder undergoes code pretraining to understand each concept in two manners: 1) Go through the class definition code of each concept. 2) Go through the instance codes of each concept. In the schema following phase, KnowCoder is finetuned using instruction tuning code, where multiple task-demanded concepts are given in the schemas, enhancing KnowCoder's ability to follow schemas and generate instantiating code accordingly.

3.1 Schema Understanding Phase

3.1.1 Training Data Generation

To enhance KnowCoder's schema understanding abilities, we construct a large-scale training dataset based on the schema library. As shown in the left part of Figure 2, the training data consists of two kinds of codes, i.e., schema definition codes and instance codes. The schema definition codes are generated based on the schema library, where we randomly sample a certain number of concepts (decided by the maximum sequence length) from the

[2] We use the Wikidata dump up to 20220704.

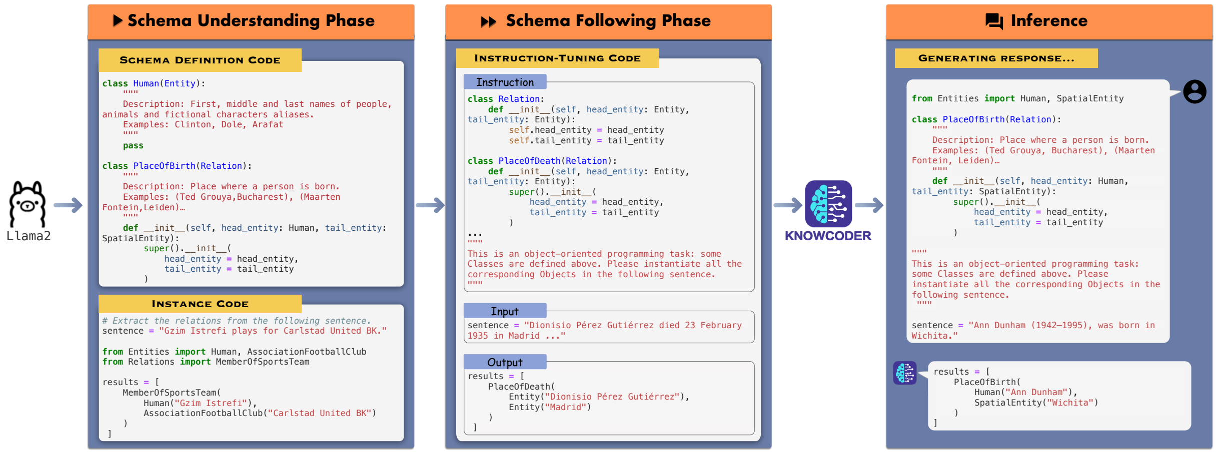

Figure 2: An diagram of training and inference processes of KnowCoder.

schema library to consist of a training sample. As the aim of the schema understanding phase is to understand each concept but not to discriminate various concepts, the instance code corresponding to a single concept contains three parts, i.e., a sentence containing instances of the given concept, an import clause to introduce the corresponding class of the given concept, and an instantiating clause to give out all the instances of the given concept in the sentence. The schema-instance codes are constructed based on KELM corpus (Agarwal et al., 2021), which contains $15,628,486$ synthetic sentences to describe the structured knowledge from Wikidata. We do data cleaning for the corpus. The cleaning details are in Appendix H.

3.1.2 Code Pretraining

After obtaining the data, we apply regular code pretraining to make LLM understand the diverse concepts in the schemas. Given a training sample with length of L, $X = x_0, x_1, ..., x_i, ..., X_{L-1}$, the model attempts to predict every token x_l based on the $x_0, ..., x_{l-1}$, where $l = 0, ..., L-1$. Some training details are as follows:

Schema Importing. The straightforward way to construct a pretraining sample is to directly give the whole schema definition for the corresponding instance code. However, this manner may cause the model to overfit the schema definition code because they are frequently repeated in every instance code. To address this problem, we separate the schema definition code from the instance code and use the "import" clause to introduce the corresponding schema definition to the instance code.

The position of the "import" clause is also critical for the LLMs to learn. We study two positions for the "import" clause, i.e., "Import-First" and "Sentence-First". We adopt "Sentence-First" because it performs better than the others. The comparison results are in Appendix A.

3.2 Schema Following Phase

3.2.1 Training Data Generation

To enhance the schema following abilities of KnowCoder, we construct instruction tuning training data for UIE tasks. As shown in the middle part of Figure 2, a typical instruction tuning sample contains three parts of codes, i.e., instruction code T, input code I, and output code O.

The instruction code T comprises two snippets, i.e., schema definition and task description. The schema definition snippet includes definitions of some concepts selected from the former phase, which defines specific concepts to be extracted. The task description snippet includes a comment that contains a natural language description of an IE task. For example, the task description of Relation Extraction (RE) is "This is an object-oriented programming task: some Classes are defined above. Please instantiate all the corresponding Objects in the following sentence.". The input I contains the sentence to be extracted, which is denoted as a variable "sentence", i.e., "sentence = ...". The output O contains all the golden knowledge in the sentence, denoted as a list variable "results", i.e., "results = [...]". We have conducted a performance comparison of different versions of the instructions, and the corresponding results are in Appendix D.

We construct the training corpus from three data sources. For Named Entity Extraction (NER), ChatGPT-annotated Pile corpus (Zhou et al., 2023) is selected. For Relation Extraction (RE) and Event Extraction (EE), we adopt the data sources constructed in Gui et al. (2023)[3] and LSEE (Chen et al., 2017), respectively.

3.2.2 Instruction Tuning

The objective of instruction tuning is to learn an LLM $\mathbf{f} : (I \times T) \rightarrow O$. The LLM takes input code I, and instruction code T as input. Subsequently, the LLM is tuned to generate every token in the output O. Some training details are as follows:

Negative Class Sampling. In the constructed schema library, there are more than 30000 concepts. It is challenging for the model to accommodate all the corresponding class definitions in a single prompt. Consequently, KnowCoder employs a negative class sampling strategy. For each training sample, in addition to the classes annotated in the sentence, we randomly sample several classes (20% number of the golden classes) from the remaining classes.

Fully negative Sample Construction. In real-world scenarios, many sentences do not contain any knowledge of a specific IE task. To handle these scenarios, we collect some sentences and ask the model to extract types that are not mentioned in these sentences, which we call fully negative samples. Specifically, we randomly sample 5% sentences. For each sentence, we replace the golden classes with five random negative classes and set the expected outputs as empty lists.

3.3 Refinement

After schema understanding and following, we obtain KnowCoder, an LLM that demonstrates strong generalization ability on unseen schemas. Additionally, based on our unified schema representation, KnowCoder can be further refined by various human-annotated datasets simultaneously. In this phase, we conduct instruction tuning based on the datasets used in previous work (Wang et al., 2023b; Sainz et al., 2023).

In different IE datasets, concepts with the same name may follow different annotation guidelines. Take "`PERSON`" for example, in MultiNERD (Tedeschi and Navigli, 2022), entities do not include the pronouns, e.g., "he" and "she", while ACE05 (Walker and Consortium, 2005) consider personal pronouns as "`PERSON`". To alleviate the problem, we add specific dataset information in the instructions to distinguish annotation guidelines for different datasets. For example, the instruction for the ACE05 dataset is "... Please instantiate all the corresponding Event Objects in the following sentence `from DATASET ACE05`.".

[3]We use the English version of the constructed data source.

4 Experiment Setup

Datasets and Metrics. We conducted experiments using 33 specific domain Information Extraction (IE) datasets, including 23 datasets for Named Entity Extraction (NER), 8 datasets for Relation Extraction (RE), 2 datasets for Event Detection (ED) and Event Argument Extraction (EAE). The detailed statistics of these datasets are in Appendix I. Among these NER datasets, following Wang et al. (2023b); Zhou et al. (2023), we take 7 datasets as the zero-shot benchmark, including 5 datasets of different domains from CrossNER (Liu et al., 2020), MIT-Movie (Liu et al., 2019) and MIT-Restaurant (Liu et al., 2019). For RE, we adopt GIDS (Jat et al., 2018) as the zero-shot dataset. Following (Sainz et al., 2023), we adopt CASIE (Lu et al., 2021) as the zero-shot ED dataset.

To balance the evaluation coverage and costs, we introduce the KnowCoder benchmark, a composite derived from existing NER, RE, and EE datasets. Under the supervised setting, a sampling strategy was developed for NER and RE tasks to maintain the distributions of original datasets and ensure the broad coverage of knowledge types. Details on the proposed strategy and comprehensive benchmark information are available in Appendix F. For the metrics, we report the span-based offset Micro-F1 following previous methods (Lu et al., 2022; Lin et al., 2020b). More details about the metrics are in Appendix G.

Implementation Details. KnowCoder is finetuned based on LLaMA2-base-7B (Touvron et al., 2023). We utilize the Megatron-LM framework (Shoeybi et al., 2019) for schema understanding. We set the context length to 2048, the learning rate to 5×10^{-6}, the global batch size to 1M tokens, and the maximum training step to 4500. For the schema following and refinement phases, we use LoRA (Hu et al., 2021) for parameter-efficient fine-tuning. We set the lora rank and lora alpha parameters to 32 and 64, respectively. The warmup ratio is set to 0.03 and the dropout ratio is set to 0.1. The learning rates for these two phases are set to 3×10^{-4}. We limit the sequence length to 4096 and set the batch size to 256. Detailed information about the training process is available in

Model	Movie.	Rest.	AI	Litera.	Music	Politics	Science	Average
LLaMA2-7B	31.0	19.6	30.8	24.1	28.0	38.7	44.1	30.9
LLaMA2-13B	32.6	25.2	37.5	36.5	37.0	60.3	**51.7**	40.1
LLaMA2-7B	31.0	19.6	30.8	24.1	28.0	38.7	44.1	30.9
KnowCoder-7B (SU. only)	**37.2**	**36.4**	**41.8**	**42.6**	**53.8**	**60.6**	51.6	**46.3**$^{\uparrow 49.8\%}$

Table 1: Results on NER under the few-shot setting.

Model	Movie.	Rest.	AI	Litera.	Music	Politics	Science	Average
w. refinement								
InstructUIE-11B (Wang et al., 2023b)	-	-	48.4	48.8	54.4	49.9	49.4	-
GoLLIE-7B (Sainz et al., 2023)	63.0	43.4	59.1	62.7	67.8	57.2	55.5	58.4
GoLLIE-13B (Sainz et al., 2023)	62.5	49.8	56.7	59.7	65.5	54.4	56.2	57.8
UniNER-7B (refined) (Zhou et al., 2023)	59.4	31.2	62.6	64.0	66.6	66.3	69.8	60.0
w.o. refinement								
Vicuna-7B (Chiang et al., 2023)	6.0	5.3	12.8	16.1	17.0	20.5	13.0	13.0
Vicuna-13B (Chiang et al., 2023)	0.9	0.4	22.7	22.7	26.6	27.2	22.0	17.5
ChatGPT (Ouyang et al., 2022)	5.3	32.8	52.4	39.8	66.6	68.5	**67.0**	47.5
UniNER-7B (Zhou et al., 2023)	42.4	31.7	53.5	59.4	65.0	60.8	61.1	53.4
KnowCoder-7B	**50.0**	**48.2**	**60.3**	**61.1**	**70.0**	**72.2**	59.1	**60.1**$^{\uparrow 12.5\%}$

Table 2: Results on NER under the zero-shot setting. ***w. refinement*** denotes methods that are refined on human-annotated data, which is unfair for KnowCoder to compare with.

Appendix K. During the inference phase, we use greedy search and set the temperature to 0. The maximum output length is set to 640.

Dataset	SoTA	7B
GIDS_{RE}	(Ouyang et al., 2022) 9.9	**25.5**
CASIE_{ED}	(Sainz et al., 2023) **59.3**†	58.2
Average	34.6	**41.9**$^{\uparrow 21.1\%}$

Table 3: Results on RE and ED tasks under the zero-shot setting. † indicates that it is unfair for KnowCoder to compare with the score.

5 Results and Analyses

5.1 Few-shot Evaluation After Schema Understanding

Considering that a pre-trained LLM cannot give proper results without given examples, we study the generalization ability of KnowCoder after the schema understanding phase, denoted as KnowCoder (SU. only), under the few-shot setting. Specifically, We utilize the first five samples from the training data as examples and report the NER F1 score in Table 1 across zero-shot NER datasets. The results demonstrate that KnowCoder (SU. only) outperforms LLaMA2-7B with an average relative improvement of **49.8%**. Remarkably, KnowCoder (SU. only) gets an average F1 of 46.3% with only a few examples, which are comparable to InstructUIE refined using human-annotated datasets. The results strongly support the effectiveness of the schema understanding phase in enhancing model generalization and performance in NER.

5.2 Zero-Shot Evaluation After Schema Following

To verify the generalization ability of KnowCoder, we conduct zero-shot experiments on 9 datasets across NER, RE, and ED tasks. In this setting, we employ KnowCoder after schema understanding and following to conduct extraction. KnowCoder is compared with two kinds of baselines. One is the LLM-based IE method that refined on human-annotated data, including InstructUIE (Wang et al., 2023b), GoLLIE (Sainz et al., 2023), and UniNER (Zhou et al., 2023). The other is models without refinement, including Vicuna (Chiang et al., 2023), ChatGPT, UniNER (Zhou et al., 2023). The results of these three baselines are from Zhou et al. (2023). Note that KnowCoder is unfair when compared with methods after refinement.

Main Results. The results of zero-shot NER are in Table 2. It can be seen that KnowCoder sur-

Ratio	Model	Task				Average
		NER	RE	ED	EAE	
1%	UIE-base	**82.8**	30.8	41.5	12.8	42.0
	LLaMA2-7B	72.3	32.1	35.3	33.3	43.3
	KnowCoder-7B	79.2	**43.3**	**50.3**	**38.5**	**52.8**$^{\uparrow 21.9\%}$
5%	UIE-base	88.3	**51.7**	55.7	30.4	56.5
	LLaMA2-7B	89.3	35.7	52.6	46.3	56.0
	KnowCoder-7B	**90.6**	51.1	**59.0**	**48.3**	**62.3**$^{\uparrow 10.3\%}$
10%	UIE-base	89.6	**59.2**	60.3	36.3	61.4
	LLaMA2-7B	91.2	48.6	60.7	52.3	63.2
	KnowCoder-7B	**92.2**	53.6	**62.2**	**55.1**	**65.8**$^{\uparrow 4.1\%}$

Table 4: Low-resource results on IE tasks, where **Average** is the average F1 across four IE tasks.

passes baselines without refinement across four NER datasets, registering a relative performance enhancement of **12.5%**. This improvement is attributed to KnowCoder's training on a large-scale, automatically generated dataset within a two-phase learning framework, which enhances its generalization capabilities for NER, even surpassing methods refined with human-annotated data. The results of zero-shot RE and ED are in Table 3. For ED, KnowCoder's performance is inferior to GoLLIE, a baseline model trained on high-quality, human-annotated data. This emphasizes that human-annotated datasets can enhance performance for more difficult IE tasks, such as ED. To further substantiate the point, we further refine KnowCoder with the ACE05 dataset, the same EE training data employed by GoLLIE. This refinement significantly improves zero-shot F1 performance to 72.0% on the CASIE dataset. This represents a significant advancement over GoLLIE of 59.3%, marking a relative improvement of **21.4%**.

5.3 Low Resource Evaluation After Schema Following

To further investigate the generalization ability of KnowCoder for IE tasks, we conduct low-resource experiments by fine-tuning KnowCoder with three different partitions of the original training sets (1/5/10% ratio) across four tasks. Following Lu et al. (2022), we adopt CoNLL03, CoNLL04, ACE05_{ED} and ACE05_{EAE} as the benchmarks for NER, RE, ED, and EAE tasks. LLaMA2 denotes directly fine-tuning LLaMA2 with these partial training data. The results are in Table 4. It can be shown that KnowCoder gets the highest average F1 scores across all IE tasks in low-resource settings at varying ratios. In ratio 1%, KnowCoder gets the relative average improvement of **21.9%**

Dataset	SoTA		7B
ACE04	(Lu et al., 2022)	**87.6**	86.2
ACE05	(Sainz et al., 2023)	**89.6**	86.1
AnatEM	(Zhou et al., 2023)	**88.9**	86.4
Broad Twitter	(Zhou et al., 2023)	**79.8**	78.3
CoNLL03	(Zhou et al., 2023)	94.8	**95.1**
DIANN	(Sainz et al., 2023)	84.1	**94.7**
FabNER	(Zhou et al., 2023)	82.3	**82.9**
FindVehicle	(Zhou et al., 2023)	98.4	**99.4**
GENIA	(Zhou et al., 2023)	**80.3**	76.7
Movie	(Zhou et al., 2023)	90.2	**90.6**
Rest.	(Wang et al., 2023b)	**82.6**	81.3
MultiNERD	(Zhou et al., 2023)	93.9	**96.1**
OntoNotes 5	(Sainz et al., 2023)	84.6	**88.2**
WikiANN	(Zhou et al., 2023)	85.4	**87.0**
WNUT17	(Sainz et al., 2023)	54.3	**66.4**
bc2gm	(Wang et al., 2023b)	80.5	**82.0**
bc5cdr	(Zhou et al., 2023)	**91.5**	89.3
ncbi	(Wang et al., 2023b)	**85.0**	83.8
Average		85.2	**86.1**$^{\uparrow 1.1\%}$

Table 5: Results on NER under the supervised setting.

compared to UIE, which shows that KnowCoder has strong adaptability to downstream IE tasks after pretraining on large-scale data under the two-phase learning framework.

5.4 Supervised Evaluation After Refinement

Under the supervised evaluation, KnowCoder is further refined with the IE datasets. We conduct supervised experiments on four IE tasks, including NER, RE, ED, and EAE. KnowCoder is compared with three kinds of methods. The first is the traditional UIE method (Lou et al., 2023; Lu et al., 2022), which is based on relatively small language models (i.e., million-level parameters). The latter two are based on LLMs (i.e., ChatGPT, LLaMA2). They adopt the in-context learning (Guo et al., 2023; Li et al., 2023; Ashok and Lipton, 2023) and supervised fine-tuning paradigms (Zhou et al., 2023; Wang et al., 2023b; Sainz et al., 2023), respectively. As some baselines only report results for specific IE tasks, we report the SOTA results of the above methods in each dataset, denoted as "SoTA" in the tables. As highlighted by Zhou et al. (2023), the evaluation script of InstructUIE (Wang et al., 2023b) contains issues. Furthermore, the benchmark in Zhou et al. (2023) remains pending release. In the end, we have implemented these two baselines on KnowCoder benchmark using their released models.

Dataset	SoTA	7B
ACE05	(Sainz et al., 2023) **70.1**	64.5
semevalRE	(Wang et al., 2023b) 65.8	**66.3**
CoNLL04	(Lou et al., 2023) **78.8**	73.3
NYT	(Wang et al., 2023b) 91.0	**93.7**
ADE corpus	(Wang et al., 2023b) 82.8	**84.3**
kbp37	(Wang et al., 2023b) 30.6	**73.2**
GIDS	(Wang et al., 2023b) 76.9	**78.0**
SciERC	(Lou et al., 2023) 37.4	**40.0**
Average	66.7	**71.7**$^{\uparrow 7.5\%}$

Table 6: Results on RE under the supervised setting.

Model	ACE05$_{ED}$	ACE05$_{EAE}$
UIE	73.4	69.3
USM	69.3	63.3
Code4UIE	37.4	57.0
InstructUIE-11B	43.2	56.8
GoLLIE-7B	72.2	66.0
KnowCoder-7B	**74.2**	**70.3**

Table 7: Results on ED and EAE under the supervised setting.

Main Results. The results for NER, RE, EE (including ED and EAE) tasks are shown in Tables 5, 6 and 7, respectively. We can observe that: (1) KnowCoder outperforms the SOTA baselines on most datasets for NER, RE, ED, and EAE, respectively. Based on the code-style schemas, KnowCoder universally models IE tasks and effectively transfers IE abilities after conducting schema understanding, following, and refinement on large-scale training data. (2) In more challenging UIE tasks, such as RE, KnowCoder demonstrates impressive advancements up to the relative improvement of 8.6% compared to the SOTA baselines. KnowCoder achieves the performances of 73.9% for ED and 66% for EAE. This is **the first time** LLM-based UIE methods surpass smaller models like UIE in ED and EAE tasks. The code-style schemas and the learning framework enable a more precise definition and understanding of this complex structured knowledge, leading to a significant improvement. (4) UniNER (Zhou et al., 2023) achieves comparable results to KnowCoder on NER. Nonetheless, KnowCoder surpasses UniNER in several respects. Primarily, UniNER is limited to extracting one type of entity per iteration, leading to a cost-time complexity. In contrast, KnowCoder can extract multiple entity types in a single iteration, enhancing efficiency. Additionally, UniNER relies on a text-style schema, making it hard to represent and extract relations and events effectively. Conversely, KnowCoder, as a UIE model, offers broader versatility and efficacy comparing to UniNER. (3) KnowCoder gets better results than baselines with code-style prompt (Li et al., 2023; Guo et al., 2023; Sainz et al., 2023). This is because KnowCoder provides a more comprehensive schema representations and conducts two-phase training to understand these schemas.

5.5 Ablation Study

To show how the schema following and understanding phases contribute to KnowCoder under the zero-shot setting, we further conduct ablation studies removing these two phases, denoted as KnowCoder (w.o. SU) and KnowCoder (w.o. SF), respectively. The results are shown in Table 8. It can be seen that: (1) KnowCoder gets better results than KnowCoder (w.o. SF) on most NER datasets. It is because the schema understanding phase helps KnowCoder to understand concepts in the schema by training on definition and instance codes and increases its generalization ability. (2) Results of KnowCoder (w.o. SF) decrease extremely, which proves the importance of schema following. Due to the lack of in-context learning ability, a 7B model without instruction tuning is hard to understand instructions under the zero-shot setting, thus making it hard to finish the IE tasks.

To demonstrate the effectiveness of the proposed KnowCoder schema, we conduct an extra ablation, denoted as "w.o. GI" in Table 8, which removes the guideline information in the schemas. The significant decline in performance verifies the importance of guideline information. Without the absence of a comprehensive schema definition, models struggle to understand concepts and cannot extract corresponding instances effectively.

6 Related Work

Universal Information Extraction. Universal Information Extraction aims to conduct different IE tasks via a single model. The existing UIE models first represent different schemas for IE tasks in a universal way. OneIE (Lin et al., 2020a) represents schemas as classification labels, InstructUIE (Wang et al., 2023b) uses keywords (Gui et al., 2023; Lou et al., 2023) of concepts to represent schemas, and UIE (Lu et al., 2022) uses a specifically-designed formal language to represent schemas. Based on

Dataset	7B	w.o. SU	w.o. SF	w.o. GI
Movie.	50.0	+1.6	**-50.0**	**-18.7**
Rest.	48.2	**-0.8**	**-46.1**	**-25.1**
AI	60.3	**-4.5**	**-57.7**	**-19.4**
Litera.	61.1	+0.6	**-59.0**	**-12.7**
Music	70.0	**-3.1**	**-69.0**	**-11.9**
Politics	72.2	**-1.8**	**-70.8**	**-12.1**
Science	59.1	**-2.7**	**-55.6**	**-12.5**

Table 8: Ablation study under the zero-shot setting.

such schema representations, these models adopt language models to understand the schemas and extract the corresponding structured knowledge.

Large Language Models for IE. Due to the strong generation abilities of LLMs, they have been used in IE recently (Xu et al., 2023). LLM-based IE methods can be divided into two categories: In-Context Learning (ICL) based methods and Supervised Finetuning (SFT) based methods. The ICL-based IE methods (Li et al., 2023; Guo et al., 2023; Ashok and Lipton, 2023; Wang et al., 2023a) make predictions only based on contexts augmented with a few examples. The SFT-based methods (Wang et al.; Gui et al., 2023; Wang et al., 2023b; Zhou et al., 2023; Xu et al., 2023; Sainz et al., 2023) use the annotated data to finetune LLMs.

Some existing work uses code-style prompts to conduct IE. Most of them are ICL-based methods. Wang et al. (2022) uses the code-style prompt to conduct event argument extraction. Li et al. (2023) uses the code-style prompt to conduct the named entity extraction and relation extraction. (Guo et al., 2023) proposes a reterive-argumented method to conduct the universal IE. These methods show relatively poor performance compared to SFT-based methods because of the lack of training to follow the schemas in the prompt. The most similar work with KnowCoder is GoLLIE, an SFT-based UIE method that gives out definitions of schemas as code comments. The difference between KnowCoder and GoLLIE is that KnowCoder designs a more comprehensive code-style schema representation method, including taxonomies, constraints, and class methods, and further constructs a large-scale schema library. Besides, GoLLIE conducts instruction tuning on human-annotated data, while KnowCoder contains a two-phase learning framework that enhances schema understanding and following ability via automatically annotated data.

Conclusion

In this paper, we introduced KnowCoder for UIE leveraging Large Language Models. KnowCoder is based on a code-style schema representation method and an effective two-phase learning framework. The code-style schema representation method uniformly transforms different schemas into Python classes, with which the UIE task can be converted to a code generation process. Based on the schema representation method, we constructed a comprehensive code-style schema library covering over $30,000$ types of knowledge. To let LLMs understand and follow these schemas, we further proposed a two-phase learning framework that first enhances the schema comprehension ability and then boosts its schema following ability. After training on billions of automatically annotated data and refining with human-annotated IE datasets, KnowCoder demonstrates remarkable performance improvements on different IE tasks under the various evalution settings.

Limitations

The schemas utilized in our approach are predominantly constructed from Wikidata, which occasionally results in some schemas lacking definitions or other relevant information. This necessitates the generation of additional data to supplement these missing elements. During the pretraining phase, we adopted a combination of automatic generation and distant supervision methods to amass a large corpus. However, this approach inevitably introduces a certain degree of noise. Furthermore, there remains room for improvement in terms of the richness and complexity of the current corpus. Further exploration of pretraining settings could also be beneficial in enhancing the zero-shot capabilities for relation and event-related tasks.

Acknowledgements

The work is supported by the Strategic Priority Research Program of the CAS under grant No. XDB0680102, the National Natural Science Foundation of China under grant No. 62306299, the National Key Research and Development Program of China under grant No. 2022QY0703, the Beijing Academy of Artificial Intelligence under grant No. BAAI2019ZD0306, the KGJ Project under grant No. JCKY2022130C039, and the Lenovo-CAS Joint Lab Youth Scientist Project. We thank reviewers for their insightful suggestions.

References

Oshin Agarwal, Heming Ge, Siamak Shakeri, and Rami Al-Rfou. 2021. Knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3554–3565, Online. Association for Computational Linguistics.

Dhananjay Ashok and Zachary C Lipton. 2023. Promptner: Prompting for named entity recognition. *arXiv preprint arXiv:2305.15444*.

Yubo Chen, Shulin Liu, Xiang Zhang, Kang Liu, and Jun Zhao. 2017. Automatically labeled data generation for large scale event extraction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 409–419, Vancouver, Canada. Association for Computational Linguistics.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Leon Derczynski, Kalina Bontcheva, and Ian Roberts. 2016. Broad Twitter corpus: A diverse named entity recognition resource. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1169–1179, Osaka, Japan. The COLING 2016 Organizing Committee.

Leon Derczynski, Eric Nichols, Marieke Van Erp, and Nut Limsopatham. 2017. Results of the wnut2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147.

Rezarta Islamaj Dogan, Robert Leaman, and Zhiyong Lu. 2014. Ncbi disease corpus: A resource for disease name recognition and concept normalization. *Journal of biomedical informatics*, 47:1–10.

Runwei Guan. 2022. Findvehicle and vehiclefinder: A ner dataset for a text-image cross-modal vehicle retrieval system.

Honghao Gui, Jintian Zhang, Hongbin Ye, and Ningyu Zhang. 2023. Instructie: A chinese instruction-based information extraction dataset. *arXiv preprint arXiv:2305.11527*.

Yucan Guo, Zixuan Li, Xiaolong Jin, Yantao Liu, Yutao Zeng, Wenxuan Liu, Xiang Li, Pan Yang, Long Bai, Jiafeng Guo, et al. 2023. Retrieval-augmented code generation for universal information extraction. *arXiv preprint arXiv:2311.02962*.

Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of Biomedical Informatics*, 45(5):885–892. Text Mining and Natural Language Processing in Pharmacogenomics.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2010. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In **SEMEVAL*.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutiérrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. *Knowledge Graphs*. Number 22 in Synthesis Lectures on Data, Semantics, and Knowledge. Springer.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Sharmistha Jat, Siddhesh Khandelwal, and Partha Pratim Talukdar. 2018. Improving distantly supervised relation extraction using word and entity based attention. *ArXiv*, abs/1804.06987.

Jin-Dong Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. 2003. Genia corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics (Oxford, England)*, 19 Suppl 1:i180–2.

Veysel Kocaman and David Talby. 2020. Biomedical named entity recognition at scale. In *ICPR Workshops*.

Aman Kumar and Binil Starly. 2021. "fabner": information extraction from manufacturing process science domain literature using named entity recognition. *Journal of Intelligent Manufacturing*, 33:2393 – 2407.

Jiao Li, Yueping Sun, Robin J. Johnson, Daniela Sciaky, Chih-Hsuan Wei, Robert Leaman, Allan Peter Davis, Carolyn J. Mattingly, Thomas C. Wiegers, and Zhiyong Lu. 2016. Biocreative v cdr task corpus: a resource for chemical disease relation extraction. *Database: The Journal of Biological Databases and Curation*, 2016.

Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023. Codeie: Large code generation models are better few-shot information extractors. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 15339–15353. Association for Computational Linguistics.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. 2020a. A joint neural model for information extraction with global features. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 7999–8009.

Ying Lin, Heng Ji, Fei Huang, and Lingfei Wu. 2020b. A joint neural model for information extraction with global features. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7999–8009, Online. Association for Computational Linguistics.

Yijin Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. 2019. GCDT: A global context enhanced deep transition architecture for sequence labeling. *CoRR*, abs/1906.02437.

Zihan Liu, Yan Xu, Tiezheng Yu, Wenliang Dai, Ziwei Ji, Samuel Cahyawijaya, Andrea Madotto, and Pascale Fung. 2020. Crossner: Evaluating cross-domain named entity recognition.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Jie Lou, Yaojie Lu, Dai Dai, Wei Jia, Hongyu Lin, Xianpei Han, Le Sun, and Hua Wu. 2023. Universal information extraction as unified semantic matching. *AAAI*.

Yaojie Lu, Hongyu Lin, Jin Xu, Xianpei Han, Jialong Tang, Annan Li, Le Sun, M. Liao, and Shaoyi Chen. 2021. Text2event: Controllable sequence-to-structure generation for end-to-end event extraction. *ArXiv*, abs/2106.09232.

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.

Alexis Mitchell, Stephanie Strassel, Shudong Huang, and Ramez Zakhary. 2005. Ace 2004 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 1:1–1.

openbiocorpora. 2015. openbiocorpora anatem.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. 2017a. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada. Association for Computational Linguistics.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. 2017b. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada. Association for Computational Linguistics.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *ECML/PKDD*.

Dan Roth and Wen tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Conference on Computational Natural Language Learning*.

Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. 2023. Gollie: Annotation guidelines improve zero-shot information-extraction. *arXiv preprint arXiv:2310.03668*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Simone Tedeschi and Roberto Navigli. 2022. MultiNERD: A multilingual, multi-genre and fine-grained dataset for named entity recognition (and disambiguation). In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 801–812, Seattle, United States. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

C. Walker and Linguistic Data Consortium. 2005. *ACE 2005 Multilingual Training Corpus*. LDC corpora. Linguistic Data Consortium.

Chenguang Wang, Xiao Liu, Zui Chen, Haoyun Hong, Jie Tang, Dawn Song, and UC Berkeley. Deepstruct: Pretraining of language models for structure prediction.

Shuhe Wang, Xiaofei Sun, Xiaoya Li, Rongbin Ouyang, Fei Wu, Tianwei Zhang, Jiwei Li, and Guoyin Wang. 2023a. Gpt-ner: Named entity recognition via large language models. *arXiv preprint arXiv:2304.10428*.

Xiao Wang, Weikang Zhou, Can Zu, Han Xia, Tianze Chen, Yuansen Zhang, Rui Zheng, Junjie Ye, Qi Zhang, Tao Gui, et al. 2023b. Instructuie: Multi-task instruction tuning for unified information extraction. *arXiv preprint arXiv:2304.08085*.

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

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. 2013. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23:170.

Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, and Enhong Chen. 2023. Large language models for generative information extraction: A survey. *arXiv preprint arXiv:2312.17617*.

Dongxu Zhang and Dong Wang. 2015. Relation classification via recurrent neural network.

Ningyu Zhang, Jintian Zhang, Xiaohan Wang, Honghao Gui, Kangwei Liu, Yinuo Jiang, Xiang Chen, Shengyu Mao, Shuofei Qiao, Yuqi Zhu, Zhen Bi, Jing Chen, Xiaozhuan Liang, Yixin Ou, Runnan Fang, Zekun Xi, Xin Xu, Lei Li, Peng Wang, Mengru Wang, Yunzhi Yao, Bozhong Tian, Yin Fang, Guozhou Zheng, and Huajun Chen. 2023. Knowlm technical report.

Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen, and Hoifung Poon. 2023. Universalner: Targeted distillation from large language models for open named entity recognition. *arXiv preprint arXiv:2308.03279*.

A Analyses on Schema Importing

To get insight into how the data organization method contributes to the results of schema understanding, we compare the performance of KnowCoder training on different version data with different schema importing methods. Specifically, we generate three versions of training data, named "Import-First", "Sentence-First", and "Whole". "Import-First" denotes that we import the class first and then give out the sentence. "Sentence-First" denotes that we import the class following the sentence, which is the version KnowCoder adopts. "Whole" denotes the version that we give out the class after the sentence with their whole definitions.

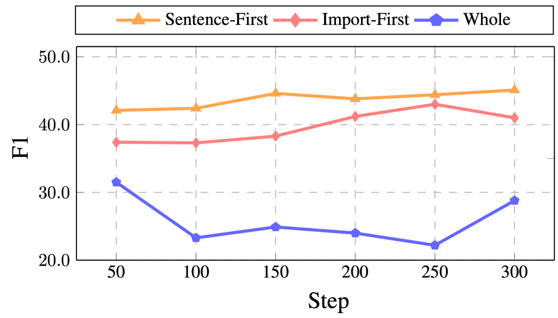

Figure 3: Detailed Analysis of different schema importing methods.

We train the model under the same setting and report the micro F1 curve on the test set of seven zero-shot NER datasets. The results are shown in Figure 3. It can be seen that "Sentence-First" performs best. If the "import" clause is before the sentence, LLMs are trained to predict the specific class after "`from Entities import`" without giving any information. The "Whole" method makes the model overfitting to the definition code because they are repeated frequently.

B Analyses on Class Name

The same concept may have different names in different IE datasets, and the concept name in downstream datasets may conflict with the name in KnowCoder schema. For example, "`Human`" in KnowCoder schema shares the same meaning as "`Person`" in ACE05. To eliminate conflicts among names of concepts in different schemas, we align the concept names in IE datasets to KnowCoder schema. Note that, for a fair comparison, we make sure the number of concepts in a dataset does not change during the alignment process. Figure 4 illustrates the F1 performance across all types under aligned and unaligned experimental settings. With average scores of 81.37 and 81.35, respectively, it can be inferred that aligning schemas does not significantly impact the model's outcomes.

C Analyses on Class Methods

Class methods are utilized to post-process the extracted results generated by LLMs. Three cases of the used class methods are listed in Table 9. To demonstrate the effectiveness of class methods, We conduct experiments on five NER datasets, including ACE05, Broad Twitter, MIT Movie, MIT Restaurant, and Ncbi-disease. The results are shown in Table 10. It can be observed that KnowCoder gets an average F1 improvement of 1%. By

Class Name	Class Method	Results	
		w.o. Class Method	w. Class Method
Average Ratings	If the extracted span is a number, add "star" after the number if "star" follows the number in the sentence.	I am looking for a unrated disney movie about a teddy bear starring julie pinson with a four star ratings average.	I am looking for a unrated disney movie about a teddy bear starring julie pinson with a four star ratings average.
Facility	Delete the content after the word "as" if "as" in the extracted span.	It lies just 12 miles from Baghdad and will be a key forward base for U.S. troops as they prepare for a push on the capital.	It lies just 12 miles from Baghdad and will be a key forward base for U.S. troops as they prepare for a push on the capital.
Organization	Delete the content after the word "such as" if "such as" in the span.	Megawati and Putin are expected to sign agreements to give Russian companies a toehold in Indonesia's oil and gas industry, long dominated by American and British giants such as Exxon Mobil and BP.	Megawati and Putin are expected to sign agreements to give Russian companies a toehold in Indonesia's oil and gas industry, long dominated by American and British giants such as Exxon Mobil and BP.

Table 9: Cases of class methods.

Model	ACE05	Bro.	Twi.	Movie.	Rest.	Ncbi.	Ave
KnowCoder-7B	85.0		77.9	90.6	81.3	82.8	83.5
+ Class Methods	↑ **0.9**		↑ **1.1**	↑ **0.9**	↑ **1.2**	↑ **0.7**	↑ **1.0**

Table 10: Results on IE tasks with Class Methods, where **Ave** is the average F1 across five datasets.

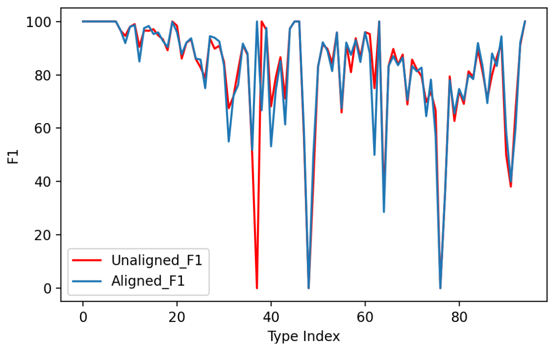

Figure 4: F1 scores of KnowCoder on each type before and after alignment for NER task.

defining some class-specific extraction rules, class methods help KnowCoder to extract more precise results.

D Analyses on Prompts

To validate the influence of different prompts on the results, Table 11 reports the performance of NER on ACE05 using prompts with two styles, i.e., Code and IE styles. It can be observed that results are similar (with a gap of 0.7% of F1), which verifies the robustness of KnowCoder to different prompts. The code-style prompt is slightly better than the IE style, suggesting that code-style prompts can better stimulate the code generation capabilities of LLMs compared to the text-style prompt and thus benefit the IE tasks.

E Analyses on Negative Sampling

To demonstrate how the negative class sampling and fully negative sample construction contribute to the results, we conduct experiments of removing the negative classes (denoted as w.o. NC) and fully negative samples (denoted as w.o. FNS), respectively. The macro average of F1 on seven zero-shot NER datasets is reported in Table 12. It can be seen that the performance of KnowCoder decreases without negative sampling, which proves the effectiveness of the negative class sampling and fully negative sample construction.

F KnowCoder Benchmark

Benchmark Construction. Considering the significant expenses associated with assessing all test sets for NER and RE tasks, we developed a sampling method to establish the KnowCoder Bench-

No.	Style	Template	F1
1	Code	Some Classes are defined above. Please instantiate the Objects corresponding to the above Classes in the sentence.	82.4
2	IE	Some Entity Types are given above. Please find all the Entities in the above Types in the sentence.	81.6

Table 11: Performance of prompts with different styles on the NER task.

Model	7B	w.o. NC	w.o. FNS
zero-shot F1	**57.8**	50.4 $^{\downarrow 7.4}$	55.7$^{\downarrow 2.1}$

Table 12: Detailed Analysis of the Negative Sampling.

Base Seed	NER	RE
1	85.0	72.5
2	84.9	71.7
42	85.3	71.6

Table 14: Results of NER and RE tasks on benchmarks with different random seeds.

mark to balance evaluation expenses and precision. Our primary principle is ensuring the sampled subset retains the same distribution. Specifically, we randomly sampled a portion of samples from each type in the dataset with a scaling factor s. For NER and RE tasks, we set s to 14 and 4, respectively. Assuming the original number of samples of the type in a dataset is x, the sampled number in the benchmark is:

$$k = \lceil x/s \rceil, s \geq 1. \tag{1}$$

Note that we adopted the same sampling method for the empty samples in datasets. Moreover, a sample may be sampled multiple times because there may be more than one type of instance. Thus, we remove duplicate samples during the sampling process. Due to the smaller number and size of EAE and ED datasets, we used the complete dataset for evaluation.

Statistics of the Benchmark. Table 13 summarizes the information on the benchmarks under the supervised setting for two tasks: NER and RE.

Task	#Sample	#Type	#Source
NER	8287	92	18
RE	5009	64	8

Table 13: Statistics of the benchmark build on NER and RE tasks under the supervised setting.

Benchmark Significance. The results reported in this paper are produced in the sampled benchmark with 42 as the base seed. To systematically assess how the generated benchmarks affect the reproducibility and consistency of the model's effectiveness, we employ multiple rounds of experiments on benchmarks with distinct random seeds, i.e., 1, 2, and 42. Table 14 summarizes the average performance on NER and RE tasks. It can be observed that the performance variations of KnowCoder across different benchmarks are minor (85.1 ± 0.2 for NER and 71.9 ± 0.5 for RE). The results demonstrate that KnowCoder's results reported in this paper are both consistent and reproducible.

G Metrics

For NER, an entity is considered correct if the entity boundary and type are correctly predicted. For RE, a relation is considered correct if its triplet matches a golden annotation, including relation type, subject entity, and object entity. For ED, an event trigger is correct if its event type and trigger match a golden annotation. For the EAE task, given an event type, an argument is correct if the argument and its role type match a golden annotation.

H Training Data Generation

The training data used in the schema understanding phase consists of two kinds of codes, i.e., schema definition codes and instance codes. In this section, we will give more details of the instance code generation process.

The instance code is generated based on the KELM corpus. The processing procedure mainly includes four steps: entity typing, entity code generation, relation code generation, and event code generation. The origin data of the KELM corpus does not annotate the types of entities. We obtain the mappings from entity names to entity types based on WikiData. Specifically, we find the corresponding WikiData ID for each entity in KELM and identify its types through the "`InstanceOf`" relations. For those entities without types, we filter them from training data. Then, we generate the entity code based on the typed KELM corpus. Finally, we clean the data by removing samples with the entity type "Wikimedia Disambiguation Page" and removing contents in brackets for entities and entity types. Based on the typed entities, we generate the relation code. Since KELM does not contain event codes, we consider relations to be events if they have sub-properties. We treat their relation types as event types, the sub-properties as corresponding role types, and the annotated mentions as arguments. Furthermore, we delete samples if the event role is one of *"of"*, *"follows"*, *"followed by"*, *"point in time"*, *"country"*.

I Data Statistics

Statistics of the Constructed Schema Library. The schema library is constructed on KELM (Agarwal et al., 2021), UniversalNER (Zhou et al., 2023), InstructIE (Zhang et al., 2023) and LSEE (Chen et al., 2017). The detailed analysis of each task schema is shown in Table 16. Here, "#Type" denotes the total number of types, "#Type w/ desc." indicates the count of types with descriptions, and "#Type w/o desc." signifies the count of types without descriptions.

Statistics of the Training Data. The training data consists of three parts: schema understanding data, schema following data, and specific domain IE data. The schema understanding training data includes schema definition codes and instance codes. The schema definition codes are built based on the schema library, with statistical results shown in Table 16. Schema instance codes are constructed based on KELM (Agarwal et al., 2021), with statistical results provided in Table 15. The schema following training data is constructed on UniversalNER (Zhou et al., 2023), InstructIE (Zhang et al., 2023) and LSEE (Chen et al., 2017). The statistics of schema following training data are presented in Table 15.

Additionally, for specific domain Information Extraction (IE), we conduct experiments utilizing 33 datasets, comprising 23 datasets for the NER task, 8 datasets for the RE task, and 2 datasets for the ED and EAE tasks. Specifically, under the supervised setting, we employ 18 datasets for the NER task, including ACE04 (Mitchell et al., 2005), ACE 2005 (Walker and Consortium, 2005), AnatEM (openbiocorpora, 2015), Broad Twitter (Derczynski et al., 2016), bc2gm (Kocaman and Talby, 2020), bc5cdr (Li et al., 2016), CoNLL03 (Sang and Meulder, 2003), DIANN (Pan et al., 2017a), FabNER(Kumar and Starly, 2021), FindVehicle (Guan, 2022), GENIA (Kim et al., 2003), MIT Movie (Liu et al., 2019) MIT Restaurant (Liu et al., 2019) MultiNERD (Tedeschi and Navigli, 2022), ncbi-disease (Dogan et al., 2014), Ontonotes5 (Weischedel et al., 2013), WikiANN (Pan et al., 2017b), and WNUT17 (Derczynski et al., 2017). For the RE task, we utilize 8 datasets under the supervised setting, including ACE 2005 (Walker and Consortium, 2005), ADE corpus (Gurulingappa et al., 2012), CoNLL04 (Roth and tau Yih, 2004), GIDS (Jat et al., 2018), kbp37 (Zhang and Wang, 2015), NYT (Riedel et al., 2010), SciERC (Luan et al., 2018), and semeval RE (Hendrickx et al., 2010). For the ED and EAE tasks, ACE05 (Walker and Consortium, 2005) and CASIE (Lu et al., 2021) are employed.

Under the zero-shot setting, we take 7 datasets for the NER task, following Wang et al. (2023b); Zhou et al. (2023), which include 5 CrossNER subsets (AI, literature, music, politics, science) (Liu et al., 2020), MIT-Movie (Liu et al., 2019) and MIT-Restaurant (Liu et al., 2019). For the RE task, we adopt GIDS (Jat et al., 2018) under the zero-shot setting. For the ED and EAE tasks, CASIE (Lu et al., 2021) is adopted under the zero-shot setting, following (Sainz et al., 2023).

The detailed statistic of each dataset is shown in Table 17. Here, "#Type" indicates the number of types, while "#Train", "#Dev", and "#Test" denote the number of sentences in the training, development, and test datasets, respectively. Figure 5 shows the overview of the datasets on specific domain IE by task and size. Note that the statistics for each dataset in the figure encompass the total number of train, dev, and test datasets.

J Details of Result Post-processing

After the output codes are generated, we obtain the extraction results based on some regular ex-

Phase	Task	Data Name	#Types	#Instance	#Tokens	Disk size	Hierarchy
Schema Understanding	NER	KELM	19,009	2,019,990	0.26B	1.15GB	✔
	RE	KELM	810	1,191,199	0.13B	0.54GB	✔
	EE	KELM	499	296,403	0.03B	0.11GB	✗
Schema Following	NER	UniversalNER	12,072	127,839	0.19B	0.96GB	✔
	RE	InstructIE	131	327,984	0.62B	2.61GB	✔
	ED	LSEE	20	415,353	0.26B	1.03GB	✗
	EAE	LSEE	20	211,635	0.10B	0.50GB	✗

Table 15: Statistics of schema understanding instance codes and schema following instruction tuning codes.

Task	#Type	#Type w/ desc.	#Type w/o desc.
NER	29,177	19,856	9,321
RE	876	840	36
EE	519	515	4

Table 16: Statistics of the constructed schema library.

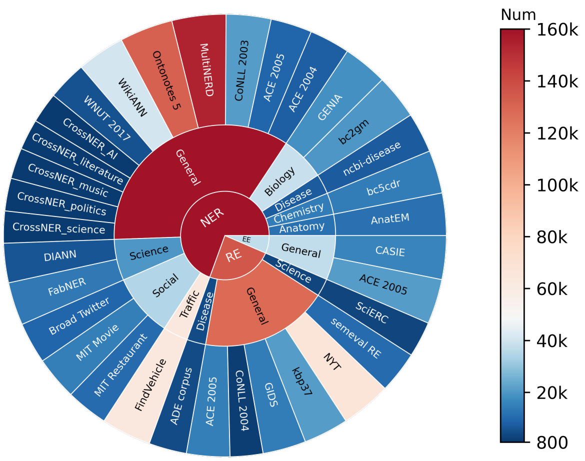

Figure 5: Overview of the datasets on specific domain IE.

pressions. To ensure the prediction results are more standardized and credible, two extra post-processing operations are added.

Superclass Induction. For the NER task, during the schema understanding phase, we have learned 29, 177 entity schemas, while the test dataset only contains 391 schemas. For specific categories, our model may provide more detailed answers which are not in the dataset schema. For example, when it comes to the entity "Harvard University", our model tends to classify it as a "`University`", while the ground truth labels it as an "`Organization`". In such cases, we employ an upper-level recursive method to address this issue. Specifically, for the predicted entity, we perform Superclass Induction based on its position in the

Task	Dataset	#Type	#Train	#Dev	#Test
NER	ACE04	7	6,202	745	812
	ACE05	7	7,299	971	1,060
	AnatEM	1	5,861	2,118	3,830
	bc2gm	1	12,500	2,500	5,000
	bc5cdr	2	4,560	4,581	4,797
	Broad Twitter	3	5,334	2,001	2,000
	CoNLL03	4	14,041	3,250	3,453
	DIANN	1	3,900	975	1,334
	FabNER	12	9,435	2,182	2,064
	FindVehicle	21	21,565	20,777	20,777
	GENIA	5	15,023	1,669	1,854
	MIT Movie	12	9,774	2,442	2,442
	MIT Restaurant	8	7,659	1,520	1,520
	MultiNERD	16	134,144	10,000	10,000
	ncbi-disease	1	5,432	923	940
	Ontonotes 5	18	107,032	14,110	10,838
	WikiANN	3	20,000	10,000	10,000
	WNUT17	6	3,394	1,008	1,287
	CrossNER_AI	13	100	350	431
	CrossNER_literature	11	100	400	416
	CrossNER_music	12	100	380	465
	CrossNER_politics	8	199	540	650
	CrossNER_science	16	200	450	543
RE	ACE05	6	10,051	2,420	2,050
	ADE corpus	1	3,417	427	428
	CoNLL04	5	922	231	288
	GIDS	4	8,526	1,417	4,307
	kbp37	18	15,917	1,724	3,405
	NYT	24	56,196	5,000	5,000
	SciERC	7	1,861	275	551
	semeval RE	10	6,507	1,493	2,717
EE	ACE05	33	19,216	901	676
	CASIE	5	11,189	1,778	3,208

Table 17: Statistics of the specific domain IE.

relationship tree in Wikidata. If the entity type of its upper-level concept matches the entity type in the ground truth, we consider the entity prediction to be correct.

Type and Text Filtering. For NER, RE, and EE tasks, if the model predicts a type that is not defined in the dataset schema and cannot be derived through superclass induction, or if an argument appears in the EAE task that is not present in the schema, we filter out such cases when calculating metrics. Additionally, if the model predicts text that does not appear in the sentence, we also filter it out.

K Implementation Details

Schema Understanding Phase. The model is trained using AdamW (Loshchilov and Hutter, 2018) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$. We set the peak learning rate to 5×10^{-6}, and use a cosine learning rate schedule with warmup ratio of 0.1, and decay final learning rate down to 10% of the peak learning rate. To mitigate overfitting, we incorporated a weight decay of 0.1 and a gradient clipping of 1.0. We configure the context length to 2048 and the global batch size to 1M tokens, with the maximum training step capped at 4500.

Schema Following Phase. We apply the LoRA (Hu et al., 2021) method to all non-embedding linear layers for schema following. During this phase, we configure the LoRA rank and alpha parameters to 32 and 64, respectively, and set a dropout rate of 0.1 to prevent overfitting. We still use the AdamW optimizer along with a cosine learning rate scheduler as in the schema understanding phase. The model undergoes 510K training samples, with a learning rate of 3×10^{-4}, a global batch size of 256, and a warmup ratio of 0.03.

Refinement Phase. In the refinement phase, we employ a parameter configuration that is largely identical to the one used during the schema following phase. However, given the richer and more varied task-type data available during the refinement stage, we opt for a greater number of training iterations. Specifically, we conduct training over three epochs, cumulatively training on 1.9M samples.

L Cases of KnowCoder Training Data

Here, we outline the cases that we have picked out from the KnowCoder-Dataset.

L.1 Instance Code in Schema Understanding Phase

NER Task

```python
# Extract the entities from the following sentence.
sentence = "Lalita Yauhleuskaya competed at the 2008 Summer Olympics."

from Entities import Human

results = [
   Human("Lalita Yauhleuskaya")
]
```

RE Task

```python
# Extract the relations from the following sentence.
sentence = "Gzim Istrefi plays for Carlstad United BK."

from Entities import Human, AssociationFootballClub
from Relations import MemberOfSportsTeam

results = [
    MemberOfSportsTeam(
        Human("Gzim Istrefi"),
        AssociationFootballClub("Carlstad United BK")
    )
]
```

EE Task

```python
# Extract the events from the following sentence.
sentence = "Jamsilsaenae station is adjacent to Sports Complex station which is on the Seoul
     Subway Line 2. The Sports Complex station is in the direction of Inner Ring Road and is
     located near Gangnam station."

from Entites import Entity
from Events import AdjacentStation

results = [
    AdjacentStation(
        connecting_line=[Entity("Seoul Subway Line 2")],
        towards=[Entity("Gangnam station")],
        direction=[Entity("Inner Ring Road")]
    )
]
```

L.2 Instruction-tuning Code in Schema Following Phase

NER Task

```
class Entity:
    """
    The base class for all entities.
    """
    def __init__(self, name: str):
        self.name = name

class Person(Entity):
    """
    Description: being that has certain capacities or attributes constituting personhood.
    Examples: patients, Donald Trump, children, women, user, patient, Trump, President Trump, Barack Obama, people
    """
    pass

class Nationality(SocialGroup):
    """
    Description: A legal identification of a person in international law, establishing the person as a subject, a national, of a sovereign state.
    Examples: American, British, Americans, German, French, English, Japanese, Russian, Australian, Indian
    """
    pass

class TvShow(Entity):
    """
    Description:
    Examples: Game of Thrones, The Walking Dead, American Idol, Modern Family, Saturday Night Live, Doctor Who, House, The Tonight Show, Mad Men, Arrested Development
    """
    pass

"""
This is an object-oriented programming task: some Entity Classes are defined above. Please instantiate all the corresponding Entity Objects in the following sentence.
"""
sentence = ‘‘I enjoyed the series ‘Professional Master Chef’ on television and I was struck by something the judges said when commenting about two of the semi-finalists. They had been highly impressed with the dishes the chefs had presented and Michel Roux Junior remarked that, despite their very obvious skill, neither chef exhibited any arrogance or conceit. Monica Galetti replied that they didn’t need to, because their work spoke for them. ’’
```

```
results = [
    TvShow("Professional Master Chef"),
    Person("Michel Roux Junior"),
    Person("Monica Galetti")
]
```

RE Task

```
class Entity:
    """
    The base class for all entities.
    """
    def __init__(self, name: str):
        self.name = name

class Relation:
    """
    The base class for all relations.
    """
    def __init__(self, head_entity: Entity, tail_entity: Entity):
        self.head_entity = head_entity
        self.tail_entity = tail_entity

class PlaceOfBirth(Relation):
    """
    Description: Most specific known (e.g. city instead of country, or hospital instead of
        city) birth location of a person, animal or fictional character.
    Examples: (Australian, London), (Muhammad, Mecca), (Augustus, Rome), (Tiberius, Rome),
        (Mozart, Salzburg), (Charles II, London), (Sima Zhao, China), (Frederick the Great,
        Berlin), (Julius Caesar, Rome), (Queen Myeongui, Goryeo)
    """
    def __init__(self, head_entity: Entity, tail_entity: Entity):
        super().__init__(head_entity=head_entity, tail_entity=tail_entity)

class Population(Relation):
    """
    Description: Number of people inhabiting the place; number of people of subject.
    Examples: (civil parish, 201), (Sao Pedro, 201), (Machame Kusini, 13,572), (Sao Joao,
        201), (unincorporated community, 15), (unincorporated community, 94),
        (unincorporated community, 25), (Mardekheh-ye Kuchek, 197), (Pain Halu Sara, 701),
        (Marenj, 1,055)
    """
    def __init__(self, head_entity: Entity, tail_entity: Entity):
        super().__init__(head_entity=head_entity, tail_entity=tail_entity)

class LocatedIn(Relation):
    """
    Description:
    Examples: (National Register of Historic Places, United States), (Ontario, Canada), (Sao
        Paulo, Brazil), (Victoria, Australia), (census-designated place, United States),
        (New South Wales, Australia), (California, United States), (Andes, Peru), (FAA,
        United States), (Norwegian, Norway)
    """
    def __init__(self, head_entity: Entity, tail_entity: Entity):
        super().__init__(head_entity=head_entity, tail_entity=tail_entity)

"""
This is an object-oriented programming task: some Relation Classes and related Entity
    Classes are defined above. Please instantiate all the corresponding Relation Objects in
    the following sentence.
"""
sentence = ‘‘Kurush is a mountain village located in the Dokuzparinsky District, in southern
    Dagestan. Situated at 2480-2560 m above sea level depending on the source , it is the
    highest continuously inhabited settlement of the Greater Caucasus and of Europe as well
    as the southernmost settlement in Russia. As of 2015, Kurush had a population of 813.’’
```

```
results = [
    LocatedIn(Entity("Kurush"), Entity("Dokuzparinsky District")),
    LocatedIn(Entity("Dokuzparinsky District"), Entity("Dagestan")),
    Population(Entity("Kurush"), Entity("813"))
]
```

ED Task

```python
class Event:
    """
    The base class for all events.
    """
    def __init__(self, trigger: str, arg_names, *args):
        self.trigger = trigger
        self.arguments = {}
        for arg_name, arg_values in zip(arg_names, args):
            self.arguments[arg_name] = arg_values

class GroupMembership(Event):
    """
    Description: Organization, club or musical group to which the subject belongs.
    Examples: singer, music, musician, play, concert, performance, singing, sang, sung, sing,
    """
    def __init__(self, trigger: str, *args):
        arg_names = ["start", "role", "end", "group", "member"]
        super().__init__(trigger=trigger, arg_names=arg_names, *args)

class OlympicMedalHonor(Event):
    """
    Description: The honor associated with winning an Olympic medal.
    Examples: medal, gold, winner, win, silver, competition, bronze, victory, player,
        compete,
    """
    def __init__(self, trigger: str, *args):
        arg_names = ["event", "country", "medalist", "medal", "olympics"]
        super().__init__(trigger=trigger, arg_names=arg_names, *args)

class Education(Event):
    """
    Description: Educational institution attended by subject.
    Examples: school, professor, coach, graduate, student, study, master, education, pupil,
        lecturer,
    """
    def __init__(self, trigger: str, *args):
        arg_names = [
            "start_date",
            "degree",
            "end_date",
            "institution",
            "student",
            "specialization",
            "major_field_of_study",
        ]
        super().__init__(trigger=trigger, arg_names=arg_names, *args)

class Marriage(Event):
    """
    Description: The subject has the object as their spouse (husband, wife, partner, etc.).
    Examples: wife, married, husband, marriage, wedding, marry, couple, spouse, mistress,
        divorce,
    """
    def __init__(self, trigger: str, *args):
        arg_names = ["spouse", "location_of_ceremony", "type_of_union", "to", "from"]
        super().__init__(trigger=trigger, arg_names=arg_names, *args)

"""
This is an object-oriented programming task: some Event Classes are defined above. Please
    instantiate all the corresponding Event Objects in the following sentence.
"""
sentence = "Thomas Lincoln on June 12, 1806 married Nancy Hanks in the Richard Berry home."
```

```python
results = [
    Marriage("married")
]
```

EAE Task

```python
class Entity:
    """
    The base class for all entities.
    """
    def __init__(self, name: str):
        self.name = name

class Event:
    """
    The base class for all events.
    """
    def __init__(self, trigger: str):
        self.trigger = trigger

class Education(Event):
    """
    Description: Educational institution attended by subject.
    """
    def __init__(
        self,
        trigger: str, # Examples: school, professor, coach, graduate, student, study, master, education, pupil, lecturer,
        start_date: List[Entity],
        degree: List[Entity],
        end_date: List[Entity],
        institution: List[Entity],
        student: List[Entity],
        specialization: List[Entity],
        major_field_of_study: List[Entity],
    ):
        super().__init__(trigger=trigger)
        self.start_date = start_date
        self.degree = degree
        self.end_date = end_date
        self.institution = institution
        self.student = student
        self.specialization = specialization
        self.major_field_of_study = major_field_of_study

"""
This is an object-oriented programming task: some Event Classes are defined above. Please instantiate all the corresponding Event Objects in the following sentence. It is important to note that the triggers of the events are confirmed as follows: "graduate" is the trigger of event type "Education".
"""
sentence = "Albert J. Herberger (born c. 1933) is a Vice Admiral of the United States Navy, and the first United States Merchant Marine Academy graduate to attain the rank."
```

```python
results = [
    Education(
        trigger="graduate",
        institution=[Entity("United States Merchant Marine Academy")],
        student=[Entity("Albert J. Herberger")]
    )
]
```