The subsequent sections constitutes the appendix.

A Additional results: Classical ML models

For comprehensiveness, we trained three classical ML models—SVM, Logistic Regression (LR), and Random Forest (RF)—using simple n-gram/tf-idf features on our causal relation classification task, for both the biomedical and general domain datasets. Please note that we trained the classical ML models using the **full training examples** rather than the 16 examples used in our few-shot experiments. The result is summarized in Table 1.

Table 1. Evaluation results with classical ML models

	`SVM`			`LR`			`RF`		
	P	R	F1	P	R	F1	P	R	F1
COMAGC	71.4	25.0	37.0	100	10.0	18.1	100	15.0	26.1
GENE	60.0	15.0	24.0	0.0	0.0	0.0	50.0	50.0	9.00
DDI	59.3	95.0	73.0	59.3	95.0	73.0	60.0	90.0	72.0
SEMEVAL	75.0	30.0	42.8	63.6	35.0	45.1	52.1	60.0	55.8

Overall, the classical ML models performed significantly poorer compared to Transformer/LM-based models (Table 2 and 3 in the main paper), even when trained on the full training data. However, it performed relatively well on the DDI datasets, achieving an F1 score of 73.0 with SVM and LR models, which outperformed the ICL model using GPT-3.5 (F1=68.9). We believe this discrepancy is due to the relatively small size of the other datasets compared to the DDI datasets, which proved that classical ML models tend to rely on larger amounts of data for effective training.

Another challenge on using classical ML approach is that it often requires complex feature engineering, while in deep learning-based approach such as Transformer-based model, the high-dimensional features are learned automatically by the "deep"-layered model, often resulting in better accuracy.

B Additional results: zero-shot prompting

We included the results of zero-shot prompting, summarized in Table 2. As in few-shot prompting (i.e., the ICL model), we used the `gpt-3.5-turbo-instruct` LLM for zero-shot prompting, as well. To summary the results, the zero-shot baselines have shown inferior performance compared to the other models (few-shot and full fine-tuning).

Table 2. Evaluation results: zero-shot prompting

	P	R	F1
COMAGC	51.7	75.0	61.2
GENE	50.0	15.0	23.0
DDI	40.0	20.0	26.6
SEMEVAL	53.3	80.0	64.0

C Baseline (`GPT-3.5-turbo-instruct`) hyperparameters

In this work, we used OpenAI[1] API with `gpt-3.5-turbo-instruct` engine. In general we assume only query access to the LLMs (i.e., no gradients, no log probabilities). Table 3 summarizes the hyperparameter values for the ICL model experiment with `gpt-3.5-turbo-instruct`.

Table 3. Hyperparameter values.

hyperparameter	value
`temperature`	0.7
`max token`	$100 \sim 400$
`top p`	1
`frequency penalty`	0
`presence penalty`	0

D MLM hyperparameters settings

We used the `biomed_roberta_base`[2] for all biomedical datasets (COMAGC, GENEC, and DDI), and `roberta-base`[3] for the SEMEVAL dataset. We used both models from the `huggingface` [1] models library. All models are implemented in Python with `Pytorch`[4] and `Transfomer`[5] library. The random seed of 203 is set for all experiments. Table 4 summarizes the hyperparameter values for fine-tuning the model experiments.

[1] https://platform.openai.com/
[2] https://huggingface.co/allenai/biomed_roberta_base
[3] https://huggingface.co/FacebookAI/roberta-base
[4] https://pytorch.org/
[5] https://huggingface.co/docs/transformers/en/index

Table 4. MLM hyperparameter values.

hyperparameter	COMAGC	GENEC	DDI	SEMEVAL
`batch size`	8	8	8	8
`max length`	256	256	256	256
`optimizer`	`Adam`	`Adam`	`Adam`	`Adam`
`lr`	3e-5	3e-5	3e-5	3e-5
`gradient acc. steps`	4	4	4	4
`adam eps.`	1e-06	1e-06	1e-06	1e-06
`warmup proportion`	0.06	0.06	0.06	0.06
`weight decay`	0.01	0.01	0.01	0.01
`max grad norm`	1	1	1	1
`epoch`	30	30	30	20

E CLM/decoder-only hyperparameters settings

In all evaluation experiments, the `bloomz-560m`[6] model from the `huggingface` models library were used for all datasets. All models are implemented in Python with `Pytorch`[7] and `Transfomer`[8] library. The random seed of 203 is set for all experiments. Table 5 summarizes the hyperparameter values for fine-tuning the `bloomz-560m` model experiments.

Table 5. `bloomz-560m` hyperparameter values.

hyperparameter	COMAGC	GENEC	DDI	SEMEVAL
`batch size`	4	4	4	4
`max length`	256	256	256	256
`optimizer`	`AdamW`	`AdamW`	`AdamW`	`AdamW`
`lr`	3e-5	3e-5	3e-5	3e-5
`gradient acc. steps`	4	4	4	4
`warmup proportion`	0.06	0.06	0.06	0.06
`max grad norm`	1	1	1	1
`epoch`	30	10	15	20

F Seq2SeqLM hyperparameters settings

In all evaluation experiments, the `t5-base`[9] model from the `huggingface` models library were used for training all datasets. All models are implemented in Python

[6] https://huggingface.co/bigscience/bloomz-560m
[7] https://pytorch.org/
[8] https://huggingface.co/docs/transformers/en/index
[9] https://huggingface.co/google-t5/t5-base

with `Pytorch`[10] and `Transfomer`[11] library. The random seed of 203 is set for all experiments. Table 6 summarizes the hyperparameter values for fine-tuning the `t5-base` model experiments.

Table 6. `t5-base` hyperparameter values.

hyperparameter	COMAGC	GENEC	DDI	SEMEVAL
`batch size`	8	4	8	8
`max length`	256	256	256	256
`optimizer`	`AdamW`	`AdamW`	`AdamW`	`AdamW`
`lr`	3e-3	3e-4	3e-4	3e-4
`gradient acc. steps`	4	4	4	4
`warmup proportion`	0.06	0.06	0.06	0.06
`max grad norm`	1	1	1	1
`epoch`	30	20	20	20

G Querying the KGs

We access the **Hetionet** KG through its official public Neo4j API[12], with `neo4j.v1` Python library. Neo4j[13] is a third-party graph database that supports the Cypher language for querying and visualizing a knowledge graph. Hetionet also provides a public Neo4j browser app[14].

As for the **Wikidata**, we access the KG through its official public SPARQL endpoint[15], with `SPARQLWrapper` Python library. We employ the official wikidata API (e.g., `wbsearchentities` and `wbgetentities` functions) for extracting the Wikidata IDs for all variable pairs.

On Hetionet, we query up to 4 hops for extracting the KGs structures. However, for Wikidata, we query up to one hops to extract the KG structures, constrained by its huge sizes. To train a robust models that generalize well given any KG structures, we opted to not optimize the content from the KG structures to be included in the prompt. For instance, when there are more than m metapaths for a pair, we randomly select m of them, m being a *hyperparameter* of the number of metapaths to be included as prompt.

In all experiments, we include up to the following: 4 neighbors nodes, 5 common neighbors nodes, and 1 metapath, to be included in the prompt. These values are selected based on hyperparameters setting giving the best results in our experiments.

[10] https://pytorch.org/
[11] https://huggingface.co/docs/transformers/en/index
[12] bolt://neo4j.het.io
[13] https://neo4j.com/
[14] https://neo4j.het.io/browser/
[15] https://query.wikidata.org/sparql

H Prompt hyperparameters and example

(to-be-added)

References

1. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., Rush, A.: Transformers: State-of-the-art natural language processing. In: Liu, Q., Schlangen, D. (eds.) Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020). https://doi.org/10.18653/v1/2020.emnlp-demos.6, https://aclanthology.org/2020.emnlp-demos.6