DACO: Towards Application-Driven and Comprehensive Data Analysis via Code Generation

Anonymous ACL submission

Abstract

Data analysis is a crucial analytical process to uncover valuable insights from the given data. In this work, we tackle this challenging task that aims at automatically generating in-depth studies and conclusive insights to comprehensively answer a given user query for tabular data. We construct the DACO dataset for this task, containing (1) 440 databases (tables) collected from real-world scenarios, (2) $\sim 2k$ answer annotations automatically generated by leveraging LLMs that can serve as weak supervision for model fine-tuning, and (3) a concentrated but high-quality human refined subset that serves as our main evaluation benchmark. For generating the automatic annotations, we leverage the code-generation capabilities of LLMs and propose a multi-turn prompting technique to automate the data analysis process for each query. Using our proposed benchmark, we additionally investigate whether the model generations and human expectations (of a good analysis) can be aligned, we design a set of dense reward models that are integrated into a reinforcement learning (RLHF) framework to train language models. Our DACO-RL algorithm is evaluated by human annotators to produce more helpful answers than supervised fine-tuning (SFT) baseline in 57.72% cases, justifying the effectiveness of the proposed algorithm.

1 Introduction

Data analysis is the process of systematically applying statistical and/or logical reasoning to evaluate and comprehend data. Existing literature has investigated answering queries about information given by structural data (*e.g.*, tables) (Chen et al., 2021a; Nan et al., 2022; Lu et al., 2023b). However, they either focus on straightforward factual retrieval or short-form entity/arithmetic resolutions for specifically given entities, while real-world data analysis can involve more complex analytical processes.

Take the scenario in Figure 1 as an example, a user is investigating potential age discrimination of a shop. To effectively answer queries such as this one, a chain of logical reasoning and interacting with the data is required. For instance, *finding 1* is inferred from analyzing age distribution within the membership data ('member' table), while *finding and suggestion 2* are derived by comparing the participants' ages during the happy hours (using both 'member' and 'happy_hour_member' tables). These rigorous quantitative analyses eventually conclude the opposite to the user's hypothesis. As valuable as the conclusive suggestions such comprehensive analysis can bring, the extensive labor-efforts, hinted by these examples, can hinder the efficiency of gaining intelligence from the data in a competitive business environment. It is thus imperative to devise a system that is able to automate the aforementioned data analysis process.

To this end, we introduce a new dataset for this challenging task, DACO, **d**ata **a**nalysis via **co**de generation. DACO is constructed from a set of diverse real-world databases associated with curated user queries. In light of the previously described labor-intensive challenge, we propose to leverage large language models (LLMs) with a multi-turn chained prompts to automatically curate the analytical answers for each query. Specifically, our designed framework employs the code generation capabilities of GPT-4 (OpenAI, 2023) for automating the statistical analysis, interleaved with its ability to interpret over the obtained quantitative results.

The DACO dataset contains 440 databases and $1,942$ associated user queries. To validate the data quality of the automatically generated samples, as well as provide a refined model benchmarking resource, we additionally perform comprehensive human annotations on a subset of 100 samples.

During the human refinement process, we empirically find that, although LLM shows reasonable analytical capabilities (and hence we are able to

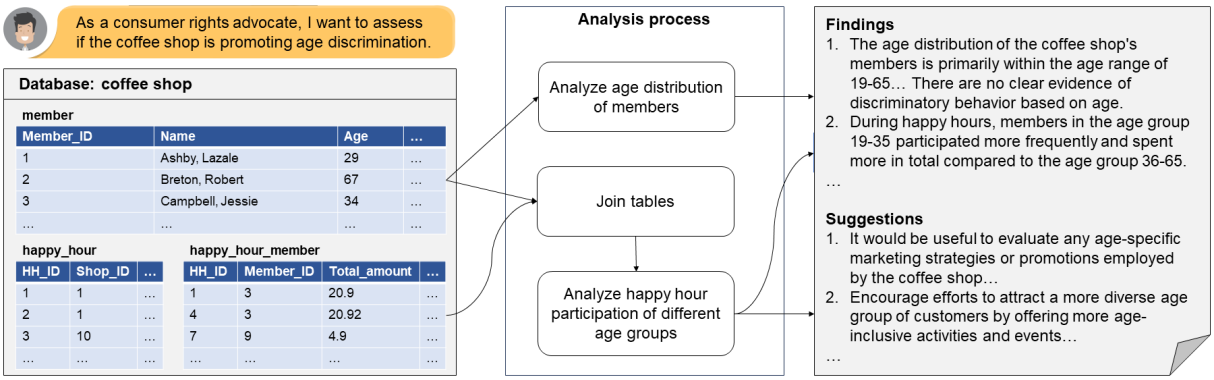

Figure 1: **Task overview.** Given an application-driven analysis query, a data analysis system is expected to produce an answer containing findings and suggestions based on the database. This requires the system to decompose the query into multiple perspectives, ground each perspective to the data, and draw conclusions from mathematical and logical reasoning. In this example, *finding 1* is inferred from analyzing age distribution within the membership data ('member' table), while *finding and suggestion 2* are derived by comparing the participants' ages during the happy hours (using 'member' and 'happy_hour_member' tables).

	Train	Dev	Test^A	Test^H	Total
# db	353	22	65	17	440
# queries	1558	100	284	100	1942
Database size			Med.	Max	Min
# tables			1	15	1
# columns			6	50	3
# rows			20	19,237	2
Answer size in Test^H			Med.	Max	Min
# findings			5	8	3
# suggestions			5	8	3
# tokens			397	864	202

Table 1: **Statistics of DACO dataset.** Train, Dev and Test^A sets are automatically generated with GPT-4, while Test^H is the human refined subset. We report the size of each data split, the size of input databases, and the size of output answers in human refined test set (Test^H).

automatically curate the pre-refined data), its generations often fall short of human expectations of what good analyses should be (*e.g.*, relevance to the queries, logical coherency, and higher-level quantitative interpretations).

In order to investigate whether aligning the models with corresponding human preferences can benefit the generations, we design a reinforcement learning algorithm (DACO-RL) that leverages two newly designed reward models (RM), *i.e.*, *contribution RM* and *regularization RM*, to efficiently provide dense feedback during intermediate model generation processes. Concretely, the contribution RM heuristically provides better learning signals for the intermediate code generation steps (for more relevant quantitative analysis), while regularization RM helps preventing reward hacking (Skalse et al., 2022) of typical RLHF models (Casper et al., 2023). We test our algorithm on a CodeGeeX-6B model (Zheng et al., 2023) (pre-trained on the pre-refined DACO data), where the win rate of 57.72% in human-annotated pairwise comparison justifies its effectiveness on learning to generate human preferred analyses.

In summary, our contributions are three folds: (1) We explore the challenging task of data analysis, where we construct the DACO dataset with our proposed multi-turn prompting technique on a diverse set of real-world databases. (2) We curate a human-refined evaluation set for benchmarking models. (3) We design the DACO-RL algorithm to jointly optimize code generation and answer generation towards human alignment, which demonstrates a significant 57.72% human evaluated win rate on the helpfulness metric of the evaluation data.

2 The DACO Task and Dataset

As shown in Figure 1, the input to our task is a database $\mathcal{D}$ and a query $\mathbf{q}$, where the output answer $\mathbf{a}$ is formatted as two lists of findings and suggestions respectively. In this work, the database $\mathcal{D}$ should be a relational database containing multiple named tables.

We construct our DACO dataset through four stages: (1) database collection, (2) query collection, (3) automatic annotation collection, and (4) human refinement. Our final dataset contains the training, development and test sets with annotations generated by GPT-4, along with a human-refined testing subset. To distinguish the two test sets, we use Test^A to represent the automatically annotated set and Test^H to represent the human refined one. Statistics are shown in Table 1.

Database collection. We collect databases from two sources: Spider (Yu et al., 2018) and Kaggle[1]. There are 157 databases collected from Spider,

[1] https://www.kaggle.com/datasets

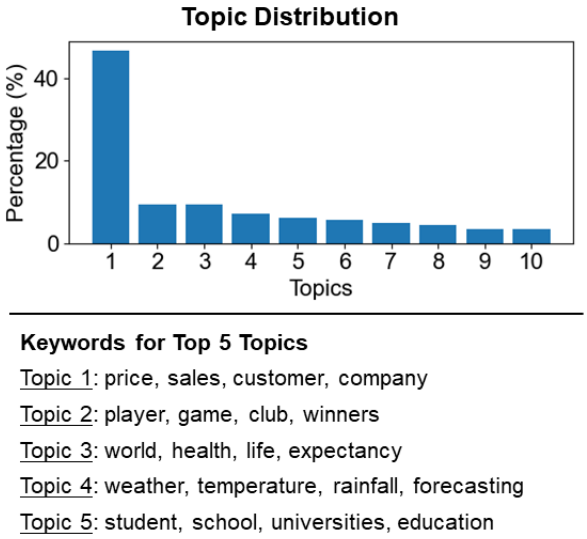

Figure 2: **Domain distribution of DACO databases.** We display the topic distribution and keywords for the leading 5 topics. Topics are extracted from database titles using BERTopic.

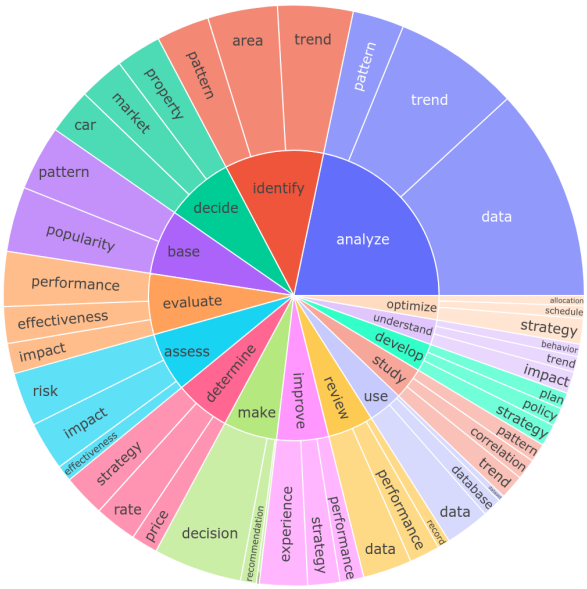

Figure 3: **Distribution of DACO queries.** We display the top 15 verbs and their top 3 direct noun objectives.

which originally come from university databases, DatabaseAnswers and Wikipedia. We additionally crawl and filter 5,830 databases from Kaggle. From this pool, we manually select a subset of 314 clean and interpretable databases to build our dataset. To maintain the diversity of the resulting database set, 157 of the databases are deliberately chosen near the long tail of its topic distribution. For this, we employ BERTopic (Grootendorst, 2022) to model the topic distribution using the database titles and descriptions, which gives in total 160 topics. We take its least frequent 80 topics as the long tail, which covers 26.79% of the total databases.

In total, DACO comprises 471 databases, each of which contains on average 2.3 tables. To better visualize the *major topic* distribution of this selected subset, we again use BERTopic but group these databases into 10 topics. The keywords for top 5 topics are shown in Figure 2. The leading topic (topic 1) is associated with business setting and consists of 46.52% of the dataset. The remaining nine topics exhibit a relatively even distribution, covering a broad range of domains, including sports (topic 2), healthcare (topic 3), weather (topic 4), and education (topic 5).

Query collection. We generate 10 queries for each database by prompting ChatGPT to first assume the role of a database stakeholder and then generate an application-driven query based on the role.

To ensure the quality of the query, we perform a manual filtering to the machine generated queries. Specifically, we remove queries that are not driven by real-world applications or cannot be answered by the given reference database. We train a group of 6 annotators to perform such a filtering process. As a result, there are about 42% of the queries removed, where the removal agreement achieves a 0.62 cohen kappa score.

After all the aforementioned processes, we obtain in total 2,664 queries. We show the top 15 verbs and their top 3 direct noun objectives in Figure 3. The queries demonstrate a notable level of diversity. The most common type of queries is to request analysis (such as "analyze data" and "identify pattern"), followed by queries aiming to make decisions (such as "determine strategy" and "make decision").

Automatic annotation collection. As shown in the right half of Figure 4, we design a pipeline that leverages the code generation capability of LLMs to automate the answer annotation for our DACO dataset. Based on the database and the query, we instruct the LLM to perform data analysis in multiple turns. At each turn, the LLM will produce a python code snippet and take its execution outputs as evidences to reason over and support its follow-up interpretation. After each turn, we prompt the model to decide whether the analysis is sufficiently comprehensive; if deemed sufficient, it terminates the coding turns and produces the final answer.

With this pipeline, we instruct GPT-4 to automatically generate all the answer annotations to each query of our dataset, for both the intermediate code and the final analysis answering the queries. To improve the quality of such automatically con-

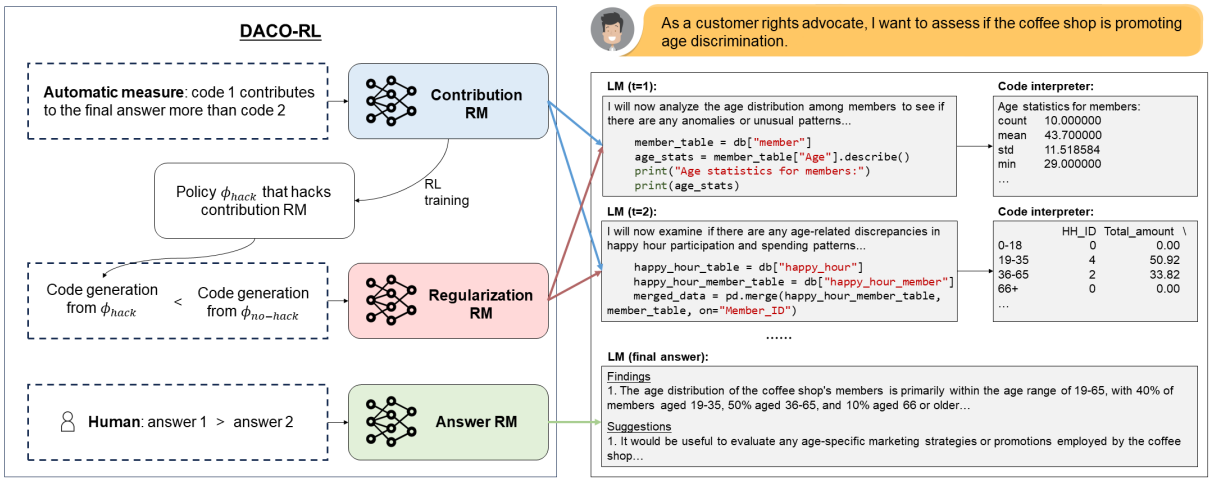

Figure 4: **Overview of code generation pipeline (right) and our DACO-RL algorithm (left).** Right: at each turn, the language model generates python code, execute the code with a code interpreter, and reads the execution outputs. Eventually, it summarizes the results into a final answer. Left: the answer RM evaluates the helpfulness of the final answer and is our end optimization goal. However, to provide denser reward, we use a contribution RM to evaluate how much each generated code contribute to the final answer. The reward is provided at the end of each generated code snippet. Contribution RM is vulnerable to reward hacking, which motivates us to propose regularization RM to discourage reward hacking generations.

structed annotations, we additionally allow GPT-4 to correct its own mistakes when its generated code leads to run-time or syntax error, where only the corrected codes are kept. We limit the number of total coding turns maximally at 9, and the number of self-correction within 2 for each turn and 4 for the whole session. In total, we obtain 1.9k valid query-answer pairs, each with roughly 3.3 intermediate coding steps.

Human refinement. The annotated analyses thus far have been algorithmically generated, where their actual quality are to be further verified. We thus curate a human-refined subset containing 100 densely human-annotated query-answer pairs. For each query, we sample 3 different analysis candidates using the previously described automated method (with GPT-4). We ask the annotators to evaluate the quality of each machine generated bullet point and categorize each point into one of bad, borderline, or good (associated with scalar scores of 0, 1 and 2 respectively, the higher the better) to each. The bullet points deemed higher quality are then mixed (from the 3 candidate analyses) and refined (with a few manual textual edits) into one final gold-analysis. In the refinement stage, the annotators should first combine all bullet points ranked as "good", remove duplicate points, and reorder the points to maintain a coherent flow. Suppose the number of bullet points are lower than our pre-defined lowest threshold (3 bullet points per answer), the annotators should select bullet points ranked as "borderline" to augment the answer. we ask a group of 3 internal members to perform refinement. The agreement accuracy of the refinement process (candidate point selection) is 0.83 and the Cohen's Kappa is 0.67.

3 DACO-RL

While DACO contains mostly algorithmic machine generated analyses, the exact reason why a second-stage human refinement process is required is to resolve potential misalignment between the machine generations and the human preferences (of "good" analyses). Our human refinement process shows that only 47.4% bullet points are "good" points perfectly addressing user queries; the majority of 52.2% are evaluated as "borderline" points that only partially aligns with human expectations; and the remaining 0.4% are considered as "bad".

We are therefore interested in investigating whether aligning human preferences via an RLHF fashion could lead to better machine generated analyses. We thus propose the DACO-RL algorithm, which is illustrated in the left half of Figure 4. Our end goal is to optimize the helpfulness of the analyzed points, which is modelled with an *answer RM* R_a. In addition to this sparse reward signal, we use a heuristically defined *contribution RM* R_c to reward each intermediate step, which is further regularized with a *regularization RM* R_r to prevent reward hacking. In the following sections, we first explain the three reward models sequentially, and eventually explain our whole RLHF pipeline.

Notations. We train a language model that interacts with the python interpreter in a conversational

manner. Formally, the full dialogue is a list of messages $[\mathbf{h}, \mathbf{c}_1, \mathbf{o}_1, \cdots, \mathbf{c}_m, \mathbf{o}_m, \mathbf{a}]$, where $\mathbf{h}$, $\mathbf{c}_i$, $\mathbf{o}_i$ and $\mathbf{a}$ stand for human message, code, execution outputs and final answer respectively. The dialogue starts with human message $\mathbf{h}$ containing both query $\mathbf{q}$ and database meta-data. In the later messages, code $\mathbf{c}_i$ and final answer $\mathbf{a}$ are generated by our model, while execution outputs $\mathbf{o}_i$ are produced by the python interpreter. To feed the dialogue into the language model, we wrap each message between a begin-of-message phrase (denoted as <BOM>) and the end-of-sentence token, and then concatenate all messages together. We use a different <BOM> for each type of message. Thus, at each turn of the conversation, the language model can decide whether to generate code $\mathbf{c}$ or final answer $\mathbf{a}$ by generating the corresponding <BOM>.

3.1 Answer RM

Our end goal is to optimize the helpfulness of answer $\mathbf{a}$, which is modelled by the answer RM R_a. Particularly, we model the helpfulness of each single bullet point rather than the full answer. We collect pairwise comparison data of bullet points from ChatGPT to train R_a. Given a pair of bullet points where one is preferred over the other, R_a is trained to assign a higher score to the preferred bullet point. Given access to the full conversation, R_a produces a reward score at the end of each bullet point in the answer.

To encourage diversity, we additionally add a repetition penalty. Given a list of findings $[\mathbf{f}_1, \cdots, \mathbf{f}_F]$, we encourage the i-th finding to be different from all previous findings by imposing a penalty score as $\sum_{j=1}^{i-1} Sim(\mathbf{f}_i, \mathbf{f}_j)$, where $Sim(\cdot, \cdot)$ computes the similarity between two bullet points.[2] This score is then subtracted from the reward score from R_a. The same procedure is applied to the suggestion list.

3.2 Contribution RM

The answer RM described above is a sparse reward signal that rewards only the answer $\mathbf{a}$ but not intermediate coding steps $\mathbf{c}_i$. To provide denser reward signal for optimizing $\mathbf{c}_i$, we aim to evaluate the helpfulness of each coding step $\mathbf{c}_i$.

However, annotating helpfulness of intermediate steps is much more difficult. For one, the helpfulness of python code is more vague to define, and the other, evaluating helpfulness requires coding expertise, which makes it more expensive to collect human annotations.

To measure the helpfulness of intermediate steps without the huge expense required by human annotations, we heuristically define the helpfulness as **how much an intermediate step contributes to the final answer**. Concretely, we compute the similarity $Sim(\mathbf{a}, \mathbf{o}_i)$ between final answer and code outputs to measure the helpfulness of $\mathbf{c}_i$. We use $Sim(\mathbf{a}, \mathbf{o}_i)$ to rank the helpfulness of different steps, and use the comparison pairs between intermediate steps to train the contribution RM R_c. Given the conversation, R_c predicts the contribution level of each $\mathbf{c}_i$ as its reward score. R_c does not take the execution output $\mathbf{o}_i$ into consideration when scoring $\mathbf{c}_i$, which simplifies model implementation and excludes spurious correlation the model may exploit from the surface form of $\mathbf{o}_i$.

3.3 Regularization RM

The heuristically defined contribution RM R_c may not necessarily perfectly align with the true helpfulness. This is due to a known reward misspecification issue termed reward hacking (Skalse et al., 2022), where the policy model achieves higher scores from the reward model but its true reward decreases. We propose to regularize such behavior with a regularization RM R_r. Given the misspecified reward model, R_c in our case, we first train an RL model until its generations start to collapse to certain patterns. These patterns typically receive high rewards from R_c but do not align well with human expectation, and thus are considered as reward hacking behaviors. We denoted this RL model as π_{hack}. We use π_{hack} to produce generations with typical reward hacking behaviors. These generations are paired with generations without reward hacking behaviors, such as generations from supervised fine-tuning (SFT) model or the pre-human refined answers generated from GPT-4, to further train the regularization RM R_r. As an intuition, this means R_r will assign lower scores to typical reward hacking behaviors.

3.4 RLHF

For our whole DACO-RL pipeline, we optimize the language model against the mixture of all three aforementioned rewards, R_a, R_c and R_r.

More specifically, we first train a multi-task reward model denoted as R_{a+c} to learn both R_a and R_c. This guarantees that the reward score distribution for R_a and R_c are relatively close, so the

[2] $Sim(\cdot, \cdot)$ is computed as the cosine similarity of SentenceBERT (Reimers and Gurevych, 2019) embeddings.

reward signal for $\mathbf{a}$ and $\mathbf{c}_i$ will not overshadow each other. We train a separate model to learn R_r, and mix R_{c+a} and R_r into $\frac{1}{2}(R_{c+a} + w_r R_r)$ when rewarding $\mathbf{c}_i$.[3] Here, w_r is a hyper-parameter to balance the variance of R_{c+a} and R_r. It is tuned to maximize the product of accuracy on the two development sets used for R_c and R_r respectively. This guarantees the mixed reward can both encourage high contribution and penalize reward hacking.

We use proximal policy optimization (PPO) (Schulman et al., 2017) as our learning algorithm. Given a Markov decision process (MDP) $\{\mathcal{S}, \mathcal{A}, P, \mathcal{R}\}$, PPO jointly optimizes a value model $V(s)$ and a policy model $\pi(s)$. The objective of the policy model is to optimize the generalized advantage estimation (GAE) (Schulman et al., 2016) $\hat{A}_t = \sum_{l=0}^{\infty}(\gamma\lambda)^l \delta_{t+l}$, where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ for each time step t. When applied to text generation, the generative language model is the policy model π, each generated token is an action, and the partially generated text sequence is the state. In our multi-turn conversational setting, however, only part of the tokens in the dialogue are generated by language model (concretely, $\mathbf{c}_i$ and $\mathbf{a}$). In other words, although the language model still takes the full conversation as input, we only compute GAE and gradients over the model generated subsequence, *i.e.*, $[\mathbf{c}_1, \cdots, \mathbf{c}_m, \mathbf{a}]$.

4 Experiments

The goal of our experiments is to verify that (1) augmenting language model with code generation can benefit data analysis, and (2) DACO-RL can further boost the answer helpfulness. To this purpose, we perform the following experiments:

Evaluated systems. We evaluate the code generation pipeline with ChatGPT and GPT-4. With the answer annotation generated by GPT-4, we further train a 6B model from CodeGeeX2-6B (Zheng et al., 2023) using both SFT and our proposed DACO-RL algorithm. For comparison, we experiment with baseline systems that directly take the linearization of raw table content as input and do not involve code generation. We experiment with zero-shot ChatGPT, zero-shot GPT-4, and CodeGeeX2-6B SFT model.

We additionally experiment with two models specifically pre-trained on tabular data, TAPAS (Herzig et al., 2020) and TAPEX (Liu et al., 2022). TAPAS (Herzig et al., 2020) is a BERT-style model pre-trained to select relevant information from a table given a user query. To apply TAPAS to our dataset, we first use TAPAS to select relevant information and then use ChatGPT to interpret the selected information. TAPEX (Liu et al., 2022) is an encoder-to-decoder model and can be fine-tuned for generation task, so we fine-tune TAPEX with our GPT-4-generated annotations.

Evaluation. For automatic evaluation, we compare the model generations against annotations (both produced by GPT-4 and curated by human annotators) and report three metrics: helpfulness, entailment, and BLEU. For helpfulness, we ask ChatGPT to evaluate whether the model generation is better than annotated answer and report the win rate. Thus, the upper bound for helpfulness is around 50. For entailment, we use an off-the-shelf NLI model to compute the probability that the model generation is entailed by the annotation. We also report the BLEU score though it does not always well reflect human preference.

We further perform two types of human evaluation, point-wise evaluation and overall evaluation. For point-wise evaluation, we ask the annotator to assign a score chosen from 0, 1 and 2 to each bullet point using the same standard as in human refinement of test set. Since the aggregation of point-wise scores may not well represent the overall quality of the generated answer, we further ask human annotators to perform pairwise comparison of two generations and select the one with a higher overall quality. Our annotation achieves high agreement of 0.65 Cohen's kappa and 0.83 accuracy for point-wise evaluation, and 0.62 Cohen's kappa and 0.82 accuracy for overall comparison.

4.1 Results

The main results are in Table 2. We have the following observations:

Code generation significantly helps data analysis, especially for zero-shot LLMs. ChatGPT and GPT-4 both enjoy a significant gain in most metrics, especially helpfulness, when equipped with code generation. As in Table 4, human evaluation further shows that GPT-4 with code generation has a significant 66.41 win rate over GPT-4 w/o code generation. After SFT, code generation brings less significant improvements, because SFT w/o code generation can simulate the behavior in GPT annotations and achieve competitive helpfulness.

[3] In practice, we observe that R_r has a large non-zero mean value m_r, where we subtract it before the weighted average.

	Method	# para.	Code gen	Test^A Help.	Test^A Entail.	Test^A BLEU	Test^H Help.	Test^H Entail.	Test^H BLEU
TableQA Baselines	TAPAS	337M	✗	**25.00**	1.96	11.62	**24.50**	**3.67**	9.73
	TAPEX	406M	✗	14.79	**3.34**	**14.60**	6.00	3.50	**13.81**
Prompt-based LLM	ChatGPT	20B†	✗	25.18	3.06	13.22	18.50	2.07	13.51
	GPT-4	175B†	✗	30.81	3.35	14.90	24.00	**4.36**	13.71
	ChatGPT	20B†	✓	35.74	2.74	14.22	27.27	2.59	14.51
	GPT-4	175B†	✓	**52.00**	**4.59**	**17.77**	**41.88**	3.26	**17.54**
Finetuned LLM	SFT	6B	✗	21.51	2.30	14.47	9.50	2.65	13.63
	SFT	6B	✓	20.95	2.15	**14.88**	11.54	4.47	**14.60**
	DACO-RL	6B	✓	**28.54**	**3.65**	13.13	**21.05**	**5.98**	11.80

Table 2: **Main results.** We report helpfulness (Help.), entailment (Entail.), and BLEU on both automatically annotated test set (Test^A) and human curated test set (Test^H). We also report the number of parameters (# para.) of each model. †: For ChatGPT and GPT-4, we report the number of parameters based on our best estimation.

	Answer Help.	Answer Help.-SFT	Answer Entail.	Answer BLEU	Code Info.-SFT	Code # lines	Code # API	Code Error
SFT	11.54	50.00	4.47	**14.60**	50.00	**41.17**	134	3.08
DACO-RL	**21.05**	**58.49**	**5.98**	11.80	**57.86**	39.66	**145**	**2.75**
w/o R_r	18.75	51.92	3.21	13.17	52.56	32.54	120	3.84
w/o R_c, R_r	8.79	40.00	3.13	11.46	44.57	14.30	108	3.91

Table 3: **Ablation study** of regularization RM (R_r) and contribution RM (R_c) in DACO-RL. We report helpfulness (Help.), entailment (Entail.) and BLEU evaluated on human curated test set (Test^H). We also use ChatGPT to compare the helpfulness of each model directly against SFT and report the win rate (Help.-SFT). For evaluating code generation, we report the informativeness win rate over SFT generations (Info.-SFT), number of code lines (# lines), number of different API (# API), and code error rate per step.

	Human point-wise	Human overall	GPT overall
GPT-4 code gen *v.s.* GPT-4 w/o code gen	**1.45**	**66.41**	**70.07**
	1.36	33.59	29.93
DACO-RL *v.s.* SFT	**1.42**	**57.72**	**58.49**
	1.30	42.28	41.51

Table 4: **Human evaluation.** We report point-wise and overall evaluation results on two pairs of models: GPT-4 with v.s. without code generation, and DACO-RL v.s. SFT. The scale of point-wise evaluation is $0 \sim 2$. We also report the overall evaluation results judged by ChatGPT which demonstrate similar trends as human evaluation.

Top 4 APIs API	Corr.	**Bottom 4 APIs** API	Corr.
print	44.24	to_datetime	-18.96
nlargest	20.06	isnull	-17.76
mean	14.56	describe	-12.02
sort_values	12.23	merge	-10.83

Table 5: **APIs** ranked by its correlation with contribution RM scores. Higher correlation means that contribution RM assigns higher scores to code snippets containing the API.

However, its mathematical and logical reasoning are not supported by code generation, so it produces more hallucination as reflected by the low entailment score on Test^H.

Our SFT model learns reasonable data analysis capabilities. By simulating GPT-4 behaviors, SFT with code generation achieves a reasonable helpfulness score and outperforms the TAPEX baseline, but still falls short compared to ChatGPT. We also evaluate the error rate (%) of generated code per step. Our SFT model has an error rate of 3.08%, which is reasonable low, but still much higher than ChatGPT (0.495%) and GPT-4 (0.491%).

DACO-RL significantly improves over SFT. As shown in Table 2 and 3, DACO-RL significantly boosts the helpfulness and entailment scores, though decreasing the BLEU score a little. Human evaluation demonstrates a 57.72 win rate of DACO-RL over SFT as in Table 4. Our qualitative analysis shows that DACO-RL better focuses on user query, while SFT tends to display generic statistics that are less relevant to user query. An example is shown in Figure 5.

We further perform ablation study and report the results in Table 3. To directly compare the ablation models against SFT, we report the win rate of each model over SFT model. To evaluate the quality of code generation, we use ChatGPT to compare the "informativeness" of generated code against SFT. The more informative analysis should produce more informative and insightful code execution outputs while staying relevant to the user query. We additionally report a few statistics including the number of code lines and the number of different API calls. We observe that without our proposed two reward models (DACO-RL w/o R_c, R_r), using only answer RM significantly hurts the model generation, leading to short and less diverse code generation and less helpful final answers. Contribution RM and regularization RM

encourages more diverse code generation and more helpful final answer production.

Contribution RM favors API calls that extracts important information from tabular data but is also vulnerable to reward hacking. We report the Pearson correlation between API occurrence and contribution RM scores in Table 5. The functions rewarded most are related to extracting significant features (`nlargest`, `sort_values`), aggregating results (`mean`), and displaying specific information (`print`). In contrast, the least rewarded functions involve displaying generic statistics (`describe`) and wrangling data (`merge`, `to_datetime`, `is_null`) since they cannot directly contribute to the user query. Examples of generated code and their contribution RM scores are shown in Figure 6a and 6b. However, we notice the concerningly high correlation between `print` function and contribution RM scores, which indicates the RL policy may exploit the correlation to hack reward. Figure 6c shows a typical reward hacking case, where the model achieves high contribution RM by printing. Our regularization RM learns to discourage such behavior and helps fix the gap.

5 Related Work

Table Question Answering. Early work in table question answering (table QA) targets simple questions that requires table lookup and cell aggregations (Pasupat and Liang, 2015; Zhong et al., 2017; Iyyer et al., 2017; Yu et al., 2018). Methods for these applications either directly applies language model to the linearized table representation (Herzig et al., 2020) or generate SQL query (Yin et al., 2020; Liu et al., 2022). Later benchmarks further require multi-hop reasoning (Chen et al., 2021a, 2020) and mathematical reasoning (Zhu et al., 2021; Chen et al., 2021b; Lu et al., 2023b). Typical solution is to chain multiple tools including table lookup and python code generation (Lu et al., 2023a). Despite the similar formulation between our task and existing table QA work, their focus are different: most existing table QA datasets focus on obtaining specific information, our data analysis queries can be complex and requires query decomposition and reasoning.

Code Generation. Code generation benchmarks have been proposed for general-purpose programming (Austin et al., 2021; Hendrycks et al., 2021), math problems (Austin et al., 2021), and data science scenario (Lai et al., 2022; Huang et al., 2022). Similar to our work, some recent work allows the language model to interact with a code execution environment and receive execution outputs as feedback (Yang et al., 2023; Wang et al., 2023). The most relevant work is Cheng et al. (2023) that also addresses data analysis via code generation. Given a data analysis query, they use GPT-4 to first generate code and then provide an interpretation of the execution results. While their analysis queries are still relatively simple, this is an early exploration aiming at automating data analysis.

RLHF. Reinforcement learning from human feedback (RLHF) aims to optimize a language model against human preference (Ouyang et al., 2022; Touvron et al., 2023; Bai et al., 2022a,b; Ziegler et al., 2019; Wu et al., 2023). While traditionally RLHF uses a holistic reward score for the entire generation (Ziegler et al., 2019; Ouyang et al., 2022), recent work shows that dense reward scores for the intermediate reasoning steps are better learning signals (Lightman et al., 2023; Wu et al., 2023). These work uses expensive human annotation to collect annotations for the dense reward data. Compared to human preference, heuristic rewards are more accessible but may not align well with true reward. This gap can lead to reward hacking (Skalse et al., 2022; Pan et al., 2022). The common remedy is to use manually designed rules to penalize behaviors that potentially harm true reward (Ouyang et al., 2022; Laidlaw et al., 2023). In this work, we directly train the reward model to discourage reward hacking behavior.

6 Conclusion

In this work, we propose a novel and challenging data analysis task, which involves decomposing user query into multiple perspectives, grounding each perspective to the input data and performing logical and mathematical reasoning. To support this task, we build the DACO dataset containing large-scale annotations automatically generated by GPT-4 and a small but high-quality test set with human curated annotations. We employ LLM enhanced with code generation to this task and evaluate three models on our dataset: zero-shot ChatGPT, zero-shot GPT-4 and a 6B SFT model. While GPT-4 consistently performs the best, SFT achieves reasonably good helpfulness with much less computation. On top of the SFT model, we further proposed our DACO-RL algorithm that significantly boosts the human evaluated helpfulness.

7 Limitations

While we put forth to construct the first of its kind dataset, DACO, for the comprehensive data analysis task, the challenging nature of the data analysis process (which often requires certain domain expertise) itself presents two major limitations of this work: **(1) It is expensive to build expert-annotated dataset.** In our work, the large-scale annotations are automatically generated by GPT-4, and their quality cannot always be well guaranteed. Although we curate a test set refined by humans, those answers are initially generated by GPT-4, which may introduce biases to human annotators during the refinement process for the final answer. Annotations curated by experts from scratch might have higher quality, but they are indeed quite costly and can create other sort of alignment problems. **(2) It is nontrivial to evaluate model generations.** Evaluating the quality of the analyses is by itself challenging and requires data science expertise. For automatic evaluation, we use ChatGPT to rank the helpfulness of model generations, which can partially but not perfectly align with human preference. Additionally, ChatGPT cannot always robustly evaluate the correctness of model generations. We use an off-the-shelf NLI model to evaluate the entailment probability between human-refined ground truths and the model generations, which can partially reflect the correctness of model generations. However, the entailment probability prediction can sometime propagate errors which lead to false positives or negatives. We make efforts to alleviate such an issue by additionally collecting human evaluations, which are supposed to better reflect the answer quality, despite that humans can occasionally exhibit subjective evaluation patterns. Notice that our annotators do not fully check the correctness of the generated answers, where we task them to focus more on the helpfulness metrics defined in this work.

8 Ethics and Broader Impacts

We hereby acknowledge that all of the co-authors of this work are aware of the provided *ACL Code of Ethics* and honor the code of conduct. The followings give the aspects of both our ethical considerations and our potential impacts to the community.

Dataset. We collect the input databases from publicly available Spider (Yu et al., 2018) dataset and Kaggle website. We further remove identity information such as emails and urls from the databases.

Potential risks. Our work aims to provide data-driven actionable suggestions to users, which may involve some critical domains including financial and medical domains. We suggest readers not to rely on suggestions generated by language model, since the reliability cannot be guaranteed.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, Benjamin Mann, and Jared Kaplan. 2022a. Training a helpful and harmless assistant with reinforcement learning from human feedback. *CoRR*, abs/2204.05862.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosiute, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemí Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022b. Constitutional AI: harmlessness from AI feedback. *CoRR*, abs/2212.08073.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Wang, Samuel Marks, Charbel-Raphaël Ségerie, Micah Carroll, Andi Peng, Phillip J. K. Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Biyik, Anca D. Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. 2023. Open problems and fundamental limitations of reinforcement learning from human feedback. *CoRR*, abs/2307.15217.

Wenhu Chen, Ming-Wei Chang, Eva Schlinger, William Yang Wang, and William W. Cohen. 2021a. Open question answering over tables and text. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. HybridQA: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036, Online. Association for Computational Linguistics.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, and William Yang Wang. 2021b. FinQA: A dataset of numerical reasoning over financial data. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3697–3711, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Liying Cheng, Xingxuan Li, and Lidong Bing. 2023. Is GPT-4 a good data analyst? *CoRR*, abs/2305.15038.

Maarten Grootendorst. 2022. Bertopic: Neural topic modeling with a class-based tf-idf procedure. *arXiv preprint arXiv:2203.05794*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333, Online. Association for Computational Linguistics.

Junjie Huang, Chenglong Wang, Jipeng Zhang, Cong Yan, Haotian Cui, Jeevana Priya Inala, Colin Clement, and Nan Duan. 2022. Execution-based evaluation for data science code generation models. In *Proceedings of the Fourth Workshop on Data Science with Human-in-the-Loop (Language Advances)*, pages 28–36, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1821–1831, Vancouver, Canada. Association for Computational Linguistics.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Scott Wen-tau Yih, Daniel Fried, Sida I. Wang, and Tao Yu. 2022. DS-1000: A natural and reliable benchmark for data science code generation. *CoRR*, abs/2211.11501.

Cassidy Laidlaw, Shivam Singhal, and Anca Dragan. 2023. Preventing reward hacking with occupancy measure regularization. In *ICML Workshop on New Frontiers in Learning, Control, and Dynamical Systems*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *CoRR*, abs/2305.20050.

Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2022. TAPEX: table pre-training via learning a neural SQL executor. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. 2023a. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023b. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, Mutethia Mutuma, Ben Rosand, Isabel Trindade, Renusree Bandaru, Jacob Cunningham, Caiming Xiong, Dragomir Radev, and Dragomir Radev. 2022. FeTaQA: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. 2022. The effects of reward misspecification: Mapping and mitigating misaligned models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1470–1480, Beijing, China. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2016. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward hacking. *CoRR*, abs/2209.13085.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2023. Mint: Evaluating llms in multi-turn interaction with tools and language feedback.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. 2023. Fine-grained human feedback gives better rewards for language model training. *CoRR*, abs/2306.01693.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. 2023. Intercode: Standardizing and benchmarking interactive coding with execution feedback. *CoRR*, abs/2306.14898.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8413–8426, Online. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *CoRR*, abs/2303.17568.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. TAT-QA: A question answering benchmark on a hybrid of tabular and textual content in finance. 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 3277–3287, Online. Association for Computational Linguistics.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul F. Christiano, and Geoffrey Irving. 2019. Fine-tuning lan-

guage models from human preferences. *CoRR*, abs/1909.08593.

Appendix

A Implementation Details

We first evaluate the performance of ChatGPT and GPT-4. We evaluate two settings, directly reading the table content, and using code generation. For the former setting, we linearize the table content into text representation as model input. Due to token limit, we feed the first 20 rows as input, which covers the full content of 93% tables. For the code generation setting, we employ the pipeline described in Figure 4. When the generated code causes a syntax or runtime error, we re-sample the model until the generated code can be executed. We allow up to 5 resamplings for each turn. We use the `gpt-3.5-turbo-16k-0613` API for ChatGPT and `gpt-4-32k` API for GPT-4.

We further finetune the CodeGeeX2-6B (Zheng et al., 2023) model for SFT and DACO-RL experiments. We first train a supervised fine-tuning (SFT) model using GPT-4 annotations, and then train our DACO-RL model on top of the SFT model. When training R_{a+c} and R_r, we initialize the model from the SFT model. When training our DACO-RL model, we initialize the value model V from R_{a+c}, and initialize the policy model π from the SFT model. In inference, we use nucleus decoding with p = 0.9 and temperature = 1.0. Similarly, we allow up to 5 resamplings when the generated code causes an error. The SFT model is trained with 8 A100 GPU for about 4 hours. The DACO-RL model is trained with 8 A100 GPU for about 18 hours. Detailed hyper-parameters are in Table 6. The only hyper-parameter we tune is λ for DACO-RL. We experiment with 0.8, 0.9 and 1.0 and discover that 1.0 works the best.

	SFT	RL
learning rate	1e-5	2e-6
gradient accumulation	4	4
total steps	600	200
λ	-	1.0
γ	-	1.0

Table 6: **Hyperparameters.**

B GPT Prompts

Prompt for query generation is in Figure 7. Prompt for helpfulness annotation collection is Figure 8. Prompt for helpfulness and informativeness evaluation is Figure 9 and 10.

C Qualitative Examples

We show final answers generated by SFT and DACO-RL in Figure 5. DACO-RL better focuses on user query, while SFT tends to display generic statistics that are less relevant to user query.

We show examples of code generations in Figure 6. We also report their reward scores from contribution RM and regularization RM.

D Artifacts

Data: We use the Spider (Yu et al., 2018) dataset distributed under CC BY-SA 4.0 license. We also use 440 publicly available datasets collected from Kaggle (`https://www.kaggle.com/datasets`) which involves multiple licenses, mostly falling under CC licenses and GPL licenses.

Software: We use transformers (Wolf et al., 2020) and deepspeed (`https://github.com/microsoft/DeepSpeed`) for model training, both distributed under Apache-2.0 license.

Model: We use CodeGeeX-2 6B (Zheng et al., 2023) for model training; `all-MiniLM-L6-v2` SentenceBERT model (Reimers and Gurevych, 2019) for computing text similarity for contribution RM; `cross-encoder/nli-roberta-base` model from transformers (Wolf et al., 2020) for entailment evaluation; `microsoft/tapex-large` and `google/tapas-base-finetuned-wtq` models from transformers (Wolf et al., 2020) as TableQA baselines; all distributed under Apache-2.0 license (except `microsoft/tapex-large` distributed under MIT license).

In summary, all the artifacts involved permit research use. Our use is consistent with their intended use. We will release the datasets and software with licenses compatible with the original access conditions. All our artifacts are limited to English and do not cover multilingual scenarios.

E Human Annotations

We perform three groups of human annotations: (1) query filtering, where we ask three internal annotators (not co-authors) to filter high-quality, application-driven and answerable user input queries; (2) test set refinement, where we ask three co-authors to refine answer annotations produced by GPT-4 into gold answers; and (3) model generation evaluation, where we ask the same three co-authors to evaluate model generation quality.

As a game developer, I want to balance the stats of different Pokemon to ensure fair gameplay.

SFT

Findings
1. There are 8 attributes (height, weight, hp, attack, defense, sp atk, sp def, speed) in the dataset that have strong positive or negative correlations, indicating that certain attributes are related.
2. The pairs of attributes with strong positive or negative correlations are as follows:
   - height and weight (correlation: 0.65)
   - hp and attack (correlation: 0.45)
   - hp and defense (correlation: 0.44)
   - hp and speed (correlation: 0.34)
3. The pairs of attributes with strong negative correlation include:
   - sp atk and defense (correlation: -0.6)
   - sp def and speed (correlation: -0.5)

Suggestions
1. For balance purposes, you could consider assigning a higher weight to taller Pokemon and a higher percentage of the total weight for Pokemon, as height and weight are strongly correlated with each other.
2. To ensure that the gameplay remains diverse and fair, you should also assign appropriate attack, defense, sp atk, sp def, and speed values to the Pokemon, taking into consideration their correlations with the other attributes.
3. Since the Pokemon attributes are strongly correlated, consider implementing a more complex system that takes advantage of this relationship, such as a network or genetic algorithm-based approach to balance the attributes of diverse Pokemon species.

DACO-RL

Findings
1. The majority of Pokemon are Water (38.13%), followed by Normal (24.72%) and Grass (13.82%).
2. Fire, Psychic, and Steel types are relatively less frequent in the dataset.
3. Types with high positive correlations with main stats (HP, attack, defense, and speed) include Water, Normal, and Grass types, while types with high negative correlations include Fire, Psychic, and Steel types.

Suggestions
1. Consider balancing the game around the most common types (Water, Normal, and Grass) to make the gameplay more diverse and interesting for different types of strategies.
2. Since types with high positive correlations have higher HPs, attacks, and speeds, it may be beneficial to focus on the Pokemon with those stats to balance the game accordingly.
3. Monitor the performance of high-positive correlation types (Water, Normal, and Grass) to ensure their representation in the game doesn't lead to an imbalance in terms of stats or gameplay strategy.
4. Monitor the performance of high-negative correlation types (Fire, Psychic, and Steel) to ensure their representation in the game isn't detrimental due to a potential imbalance in game mechanics.

Figure 5: **Case study.**

All participants are aware of the intended use for the data.

I will examine the relationships from the correlation matrix, including the most attended concerts and the conductor's experience, as well as the orchestras with the highest official ratings and their conductors.

```
# Get the top 3 most attended concerts
top_attended = merged_df.nlargest(3, "Attendance")

# Get conductor names and age with most attended concerts
top_conductors = top_attended[["Name", "Age", "Year_of_Work", "Attendance"]].sort_values("Attendance",
ascending=False)

print("Top 3 Conductors with Most Attended Concerts:")
print(top_conductors)
print()

# Get top 3 orchestras with the highest official ratings
top_rated_orchestras = merged_df.nlargest(3, "Official_ratings_(millions)")

# Get conductor names and orchestras with the highest official ratings
top_rated_conductors = top_rated_orchestras[["Orchestra", "Name", "Age", "Year_of_Work",
"Official_ratings_(millions)"]].sort_values("Official_ratings_(millions)", ascending=False)

print("Top 3 Orchestras with Highest Official Ratings and Their Conductors:")
print(top_rated_conductors)
print()
```

Contribution RM: 5.05 Regularization RM: 5.15 Two RMs combined: 5.10

(a) A good case that receives high scores from both contribution RM and regularization RM.

Next, I will join the tables 'Order_Deliveries', 'Actual_Orders', 'Products', and 'Customers' to create a single DataFrame with the necessary information to analyze the most popular products across delivery routes.

```
# Join tables
orders_deliveries = db["Order_Deliveries"]
actual_orders = db["Actual_Orders"]
products = db["Products"]
customers = db["Customers"]

# Merge tables
merged_df = pd.merge(orders_deliveries, actual_orders, on="actual_order_id")
merged_df = pd.merge(merged_df, products, left_on="actual_order_id", right_on="product_id")
merged_df = pd.merge(merged_df, customers, left_on="driver_employee_id", right_on="customer_id")

# Display a sample of 5 rows
print(merged_df.head())
```

Contribution RM: -2.63 Regularization RM: 6.59 Two RMs combined: 1.98

(b) A bad case that receives low score from contribution RM and high score from regularization RM.

Based on the analysis, provide a conclusion summarizing the findings and suggesting possible reasons for the high resettlement rates in the top 5 countries.

```
print("Conclusion:")
print("The average resettlement rate in the top 5 countries is 593.8%, which is quite high compared to the
average resettlement rate of 553.8% in the entire dataset.")
print("This is due to the high resettlement rates in the top 5 countries, which may be due to the following
reasons:")
print("1. The top 5 countries have a higher population, which may indicate a higher demand for migration.")
print("2. The top 5 countries have specialized in Industry or Commerce, which might require a high number of
workers to be migrated.")
print("3. The top 5 countries have a significant history of migration, which may indicate a diverse population
and a need for migration.")
```

Contribution RM: 8.35 Regularization RM: -5.87 Two RMs combined: 1.24

(c) A reward hacking case that receives high score from contribution RM and low score from regularization RM.

Figure 6: **Qualitative examples** of code generations, and their scores assigned by reward models.

I have a database of [database title]. I am a stakeholder and I am analyzing the database to make a decision. Who am I and what decision might it be? List 10 possibilities in a numbered list.

Each point should introduce who I am and briefly explain my intention in this format: As a/the [who I am], I want to [explain my intention]

Examples:

Based on the extracurricular activities database:
1. As the dean of student affairs, I want to decide on extracurricular activities to promote or cut
2. As the department head, I want to decide on faculty advisor assignments
3. As the school administrator, I want to review and revise faculty activity engagement

Based on a diabetes database:
1. As a healthcare policy maker, I want to decide on healthcare resource allocation
2. As a NIH official, I want to decide on medical research funding
3. As a health insurance actuary, I want to improve health insurance pricing strategy
4. As a health provider, I want to decide on patient care and treatment

Based on an allergy database:
1. As a catering manager, I want to plan meal options
2. As the school principal, I want to plan allergy awareness programs
3. As an administrator in the Student Affairs or Housing department, I want to decide on housing assignments
4. As the school administrator, I want to improve campus emergency preparedness
5. As the school principal, I want to develop policies for allergy accommodations

Based on a Home Equity Line of Credit (HELOC) product database, you can:
1. As the credit risk manager, I want to modify the credit underwriting policy

The database is as follows:

Database `[title]` has [x] tables. Table names are: [aaa], [bbb], [ccc]

Table `[caption]` has [x] rows and [y] columns. Column are:
`[column name]`, example values: [value 1], [value 2], [value 3], [value 4], [value 5]
...

Figure 7: **Prompt for query** collection.

I have a database of [database title]. As a [stakeholder role], I want to [describe intention].

Given below two findings/conclusions, which one is more helpful to my analysis?
* [answer bullet point 1]
* [answer bullet point 2]

Your response should be in the following format:
* Reasoning: <explain your reasoning here>
* Answer: <repeat the more helpful finding here>

Figure 8: **Prompt for helpfulness** annotation collection.

I have a database of [database title]. As a [stakeholder role], I want to [describe intention].

I have hired two data analysts to perform the analysis, and they gave me two different reports (listed below). Each report consists of two lists, one for findings and one for suggestions. Which one is more helpful to my analysis? When evaluating helpfulness, you should consider the following three rubrics in decreasing priority: (1) relevance to my analysis goal; (2) insightfulness; and (3) diversity of perspectives, especially for suggestions.

Your response should be in the following format. Note: <answer> should be either Report-1 or Report-2
* Answer: <answer>
* Reasoning: <explain your reasoning here>

The reports are as follows:

Report-1

[report 1]

Report-2

[report 2]

Figure 9: **Prompt for helpfulness** evaluation.

I have a database of [database title]. As a [stakeholder role], I want to [describe intention]. Below are the intermediate steps of their analysis. Which analysis is more informative? The more informative analysis should produce execution results that stick relevant to my analysis goal and bring more insights to my analysis.

Your response should be in the following format. Note: <answer> should be either Analysis-1 or Analysis-2
* Answer: <answer>
* Reasoning: <explain your reasoning here>

The reports are as follows:

Analysis-1

[analysis 1]

Analysis-2

[analysis 2]

Figure 10: **Prompt for informativeness** evaluation.