Do LLMs Paint Within the Lines? Evaluating the Ability of LLMs to Follow Fine-grained Text Editing Instructions

Anonymous ACL submission

Abstract

Instruction-tuned large language models (LLMs) have shown improved performance on a variety of NLP tasks and are used extensively in many NLP applications, including writing assistants. However, little is known about their ability to consistently and rigorously follow fine-grained instructions, especially text editing instructions. In this work, we comprehensively characterize the ability of popular LLMs to follow fine-grained text editing instructions. By introducing a benchmark suite that enables a controlled evaluation, we show that state-of-the-art LLMs show varied performance and can struggle on even elementary text editing tasks revealing key insights into the limitations of current LLMs. We finally show that further instruction tuning on text-editing instruction data can be an effective approach to improve performance on both seen and unseen text-editing tasks.

1 Introduction

Modern large-scale language models (LLMs) have been aligned to follow instructions. Such LLMs have shown the ability to perform several tasks like sentiment classification, question answering, text summarization, machine translation, and code generation when these tasks are described as instructions in suitable prompts (Wang et al., 2018, 2019; Brown et al., 2020; Hendrycks et al., 2020; Surameery and Shakor, 2023).

Despite instruction-tuned LLMs being used very widely, we do not have a clear understanding of their instruction-following ability. Prior work on rigorously evaluating the instruction-following ability of LLMs is relatively scarce (Webson and Pavlick, 2022; Min et al., 2022; Kung and Peng, 2023; Li et al., 2023). Kung et al. (2023) suggest that some instruction-finetuned models might be relying less on the instructions than previously thought. Instead, they latch onto superficial artifacts, such as the outputs' format. Li et al. (2023) argue that prior knowledge encoded in the pretraining stage can limit the steerability of models. Still, a fundamental question remains unanswered: *How good are current LLMs at following instructions? How does their effectiveness vary with the number of instructions, their complexity, and the dependence relations between instructions?*

In this paper, we seek to advance research on these specific questions in the domain of text-editing instructions. Characterizing the ability of LLMs to follow a set of instructions is critical to many practical text-editing applications in which multiple operations need to be performed in a single interaction. For example, one might want to correct formatting and grammar as well as improve clarity and then rewrite text to a specific tone. More generally, improving the ability of LLMs to follow a set of (complex) instructions would enable LLMs to rewrite text according to specific style guides, such as AP or MLA. These style guides, which contain instructions on how text must be written to conform to that style, are adopted by publishers, businesses, and media houses to ensure consistency across their communications.

We empirically investigate the above questions for popular LLM models. We introduce a suite of text-editing tasks and associated benchmarks to probe the instruction-following ability of each LLM. To do this, we identify three key dimensions on which to probe model performance. We then design text-editing tasks that assess their performance in a controlled manner across representative points along these dimensions (see Figure 1). We observe that model performance varies markedly on these tasks, and even the best models can struggle on some instructions. Our main findings are:

- LLMs find many text-editing tasks (even some elementary ones) challenging. Performance is negatively correlated with both instruction complexity and input length. Introducing de-

pendencies between instructions (especially sequential ones) poses even more challenges.
- OpenAI models generally outperform LLAMA2 models. Larger models generally perform better, which reinforces the importance of model scaling.
- Fine-tuning of models using text-editing instructions significantly improves performance over the baseline on both seen and unseen text-editing tasks, suggesting the effectiveness of further domain-specific instruction tuning.

To conclude, our analysis reveals important insights into the instruction-following ability of LLMs in the domain of text editing - an ability that is at the heart of writing assistants.

2 Setup

First, we define a set of primary (main) text editing tasks and how we constructed benchmarks for them. Next, because we also consider settings where LLMs may be fine-tuned on instructions from the main set of tasks, we also outline a set of additional text-editing tasks to evaluate the ability to generalize to new tasks/instructions in such settings. Finally, we conclude by describing the details of the evaluation strategy.

2.1 Main Tasks

Our main set of text editing tasks falls into two groups: Elementary text editing and Practical text editing. (A summary is in Figure 1):

2.1.1 Elementary Text-Editing Tasks

Elementary text-editing tasks involve elementary string manipulation. Each instruction is very simple, the expected output is unambiguous, and evaluation data can be constructed computationally.

Copy A list of instructions asking the LLM to output a specific sequence of tokens. This task probes model performance when: (a) each instruction is trivial in complexity and (b) each instruction is completely independent of others. An example is shown below:

```
Execute the following instructions:

<instruction> Output the sequence of
tokens enclosed between <token> and </token>
on a new line: <token>cat</token>
<instruction> Output the sequence of tokens
enclosed between <token> and </token> on a
new line: <token>house</token>

Output:
```

We consider two settings **Copy-1** and **Copy-N**, corresponding to single and multiple instructions.

Edit A list of instructions that transform a source string to a target string in terms of only INSERT/DELETE/REPLACE instructions on indices of the source. Such instructions can be efficiently computed using the classic Fisher-Wagner algorithm. An example of instructions that transforms `coastal` to `postal` is shown below:

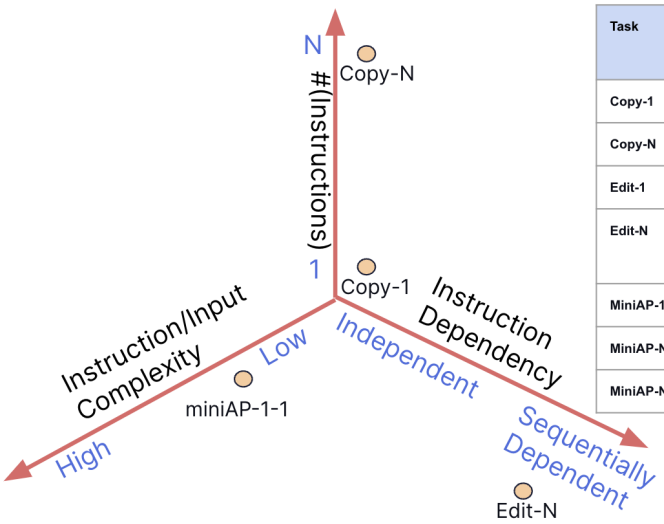
```
Execute the below instructions.
All position indexes mentioned are 0-based.

<instruction> Set s to coastal </instruction>
<instruction> Replace the character at
position 0 of s with p and set s to the new
string obtained. </instruction>
<instruction> Delete the character at
position 2 of s and set s to the new string
obtained. </instruction>
<instruction> Output only the string s
</instruction>

Output:
```

In contrast to **Copy** task, even though each individual instruction is simple, there is a *strict sequential dependency* on instructions. Each instruction operates on the output of the prior instruction. Again, we consider two settings **Edit-1** and **Edit-N**, corresponding to single and multiple instructions.

For both tasks, we generate a test (N:1000) and a non-intersecting training set (N:10000) with varying task parameters (eg. length of strings).

2.1.2 Practical text-editing tasks (miniAP)

In addition to the elementary tasks, we consider instructions on text editing and manipulation mentioned in popular language style guides, thus mirroring practical text-editing scenarios. Since we draw heavily on instructions from the AP (Associated Press) style guide, the task is called **miniAP**. We consider a set of 15 instructions (see Appendix E) in the AP style guide that are related to the usage of (a) Punctuation, (b) Abbreviations, (c) Capitalization, (d) Plurals, (e) Number formatting, and



Task	#(Inst)	Instruction Complexity	Input Complexity	Instruction Dependency
Copy-1	1	Very Low	Word	Independent
Copy-N	N	Very Low	Words	Independent
Edit-1	1	Low	Word	Independent
Edit-N	N	Low	Word	Sequential (Order Sensitive)
MiniAP-1-1	1	Medium	Sentence	Independent
MiniAP-N-1	N	Medium	Sentence	Independent
MiniAP-N-P	N	Medium	Paragraph	Independent

Figure 1: Our framework for visualizing the different dimensions on which instruction following performance can be measured. Our tasks are designed to probe model performance at different points in the space spanned by these dimensions. Task names follow the convention:`task-#(instructions)-input size`.

(f) Date and Time formatting. We follow instructions related to these dimensions because they are relatively unambiguous; compliance can be computationally verified, and collecting relevant data is easy. Instructions related to other dimensions, like clarity, tone, and sensitivity, tend to be subjective and require human evaluation, which we leave to future work. We give the model instructions from the above set, an input text, and ask the model to rewrite the text to comply with the instructions. The example below illustrates this:

```
Rewrite the text enclosed between <QASD> and
</QASD> so that it complies with the provided
style guide.
<styleguide>
<instruction> Write single-digit numbers in
words. </instruction>
</styleguide>

<QASD>
I bought 9 apples.
</QASD>

Output:
```

We look at three settings with rising complexity:

- **miniAP-1-1**: one instruction and one input sentence. This is the easiest configuration.
- **miniAP-N-1**: multiple instructions and one input sentence. This is a slightly more challenging setting since the majority of the instructions may not apply to the input sentence. Models need to ignore inapplicable instructions but still apply one or more applicable instructions.
- **miniAP-N-P**: multiple instruction with the input embedded in a five-sentence paragraph. This extends the multiple instruction setting in that the input text is larger and has multiple (potentially) non-compliant sentences [1].

To generate test and training data, we use the popular SELFINSTRUCT paradigm (Wang et al., 2022). We use GPT-4 to generate candidate sentences related to a linguistic dimension (e.g. Number usage). Then we use regular expressions to manipulate the sentences in order to generate compliant and non-compliant versions. These are manually reviewed for correctness). The is a dataset containing triplets of the form (`a style guide instruction, a non-compliant sentence, a compliant version`) and corresponds to the **miniAP-1-1** setting. To construct datasets for the other two settings, we adapt the dataset as follows: (a) for the **miniAP-N-1** setting, replace the instruction field so that it contains all instructions. If needed, we manually edit the compliant version to ensure global compliance (compliance with all instructions). (b) for the paragraph-level dataset, **miniAP-N-P**, we randomly sample records and concatenate them to generate a five-sentence paragraph. For each setting, we generate a test set (N:1000) and a training set (N:10000), ensuring that the sets are disjoint. Finally, to evaluate the ability of models to not make erroneous edits to already compliant input,

[1] Both **miniAP-N-1** and **miniAP-NP** settings resemble a realistic use case where a user might provide several instructions from a style guide and ask that the LLM rewrite input as necessary to comply.

all datasets contain some records where the input text already conforms to the instructions so no edits are needed.

2.2 Additional Tasks and Data

In addition to the main editing tasks introduced in the prior section, we outline two additional tasks. These are used to evaluate the ability of text-editing instruction fine-tuned LLM models to generalize to new instructions/tasks. We also describe the construction of an additional text instruction following dataset GENERIC-TE which we use for further fine-tuning of the LLMs.

miniAP-reversed Task We construct this set of instructions from the **miniAP-1-1** task by negating the original instructions (where the negation is unambiguous) and swapping the target and the input. This asks the model to execute the reverse of the original instruction. This strategy is applied to the **miniAP-1-1** data and is called **miniAP-reversed**.

UpperLower task (UL) Task The task and settings are identical to the **Edit** task with one major change: Instead of instructions that insert/delete/replace characters at specific indices and have sequential dependencies, the instructions involve only upper-case/lower-case characters at specific indexes. Furthermore, the final output does not depend on the instruction application order.

GENERIC-TE Dataset GENERIC-TE is a large-scale diverse text-editing instruction dataset. Prior works like ALPACA (Taori et al., 2023) have demonstrated that instruction fine-tuning on a large diverse instruction set improves model performance on various downstream NLP tasks. Building on this observation, we explore the effect of instruction fine-tuning on a large, diverse set of text-editing instructions. We construct a dataset of 50K examples of instruction following data largely adopting the process outlined in (Taori et al., 2023) with one minor change – in the SELFINSTRUCT stage, we ask the model to only focus on text-editing instructions spanning a very diverse set of linguistic dimensions (like tone, sensitive language, etc.). We use this data only for supervised fine-tuning and not for evaluation (since this dataset spans linguistic dimensions where evaluation is largely subjective).

2.3 Evaluation Details

Prompting Strategy We encode the instructions and the input text in a reasonable prompt (like the examples shown above), accommodating minor variations that may be needed (e.g. removing the system prompt, etc.). We adopt a best-effort approach to prompt engineering, incorporating known best practices to design prompts. An implicit but practical assumption is that "prompt engineering effort" is negatively correlated with "model steering/instruction-following ability". Therefore, when a model requires extensive prompt engineering to follow instructions, intuitively, it implies the model is not as steerable and ranks lower than a model that requires less effort.

Evaluation Metric Because we want to measure the ability of models to follow text editing instructions precisely, we use an exact match with the compliant text to be a measure of success and to report accuracy. To maximize success potential, we allow models to be flexible on the output format and perform appropriate post-processing on the output as long as they are largely consistent with their output format (e.g., some models output rewritten text between `<output>` `</output>` tags while some just output the rewritten text).

3 Models

We consider three models: ChatGPT, GPT4 (OpenAI, 2023), and LLAMA2 (Touvron et al., 2023). ChatGPT and GPT-4 are closed models, but they are considered to be state-of-the-art and thus establish strong baselines. However, because these models are closed, to investigate the effect of aspects like (a) model scale, (b) training procedures, and (c) training data, we consider LLAMA2 an open model as well. We focus on the 13B variant unless specified. [2]

4 Zero-shot Performance

We report and analyze the zero-shot performance of our models on our established tasks.

Overall Performance. Figure 2 shows overall performance from which we note:

- All models perform the **Copy** task almost perfectly. Recall that this task involves instructions where (a) each instruction is trivial in complexity and (b) instructions are completely independent. **This suggests that very low**

[2] We consider the chat variant of LLAMA2 since the base model is not instruction fine-tuned. We focus on the 13B variant since it is both an expedient choice and also more representative of practical use cases.

instruction complexity and independence correlate with higher performance.

- All models find the **Edit** task challenging. Even performance on the single instruction setting (**edit-1**) is low. Prior work has noted that LLMs (including GPT4) may still struggle at seemingly elementary tasks like counting, article swapping, and shift ciphers (McCoy et al., 2023). Index-based string editing operations is potentially yet another task. Finally, performance drops even further in the multiple-instruction setting, suggesting that **sequential dependencies between instructions pose additional challenges.** We conjecture that this is because models need to operate implicitly on intermediate outputs, thus introducing more points of failure.
- On the **miniAP** tasks, we note that performance on the single instruction, single sentence setting (**miniAP-1-1**) is generally higher than the other settings (**miniAP-N-***) revealing a clear trend that **model performance decreases as input size increases**. This observation further supports the observation around general challenges of LLMs handling long contexts (Liu et al., 2023). We characterized how performance drops as input length increases by analyzing performance at first P sentences shown in Figure 3.
- **OpenAI models outperform LLAMA2 on average.** Both ChaGPT and GPT4 generally outperform LLAMA2. Between ChatGPT and GPT4, GPT4 generally performs similarly to ChatGPT except for the **Edit** task, where it significantly outperforms ChatGPT.

Effect of Model Scale. Figure 4 shows the impact of model scale on performance. We can draw the following conclusions: On very simple tasks, like the Copy task, smaller models perform as well as larger ones. On tasks that involve non-trivial instructions, the 7B and the 13B models generally perform similarly. The largest model (70B) shows a clear benefit overall (especially on the **edit-1**, **miniAP-1-1** and **miniAP-N-1** tasks), suggesting that model scaling might yield improvements beyond specific thresholds (here, beyond 13B).

Effect of Instruction Alignment with Prior Knowledge. Because LLMs are trained on massive amounts of text and instructions from diverse

[3]Error bars represent 95% confidence intervals.

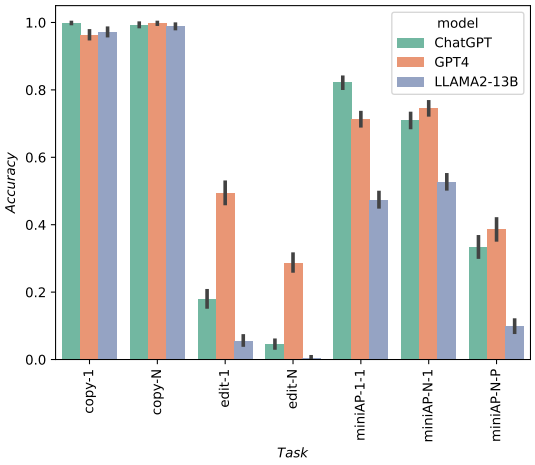


Figure 2: **Overall performance of models on following text editing instructions**. Models achieve relatively high accuracy when asked to follow a single instruction or when the instructions are very simple. However, they struggle in following multiple instructions, or on larger inputs. [3]

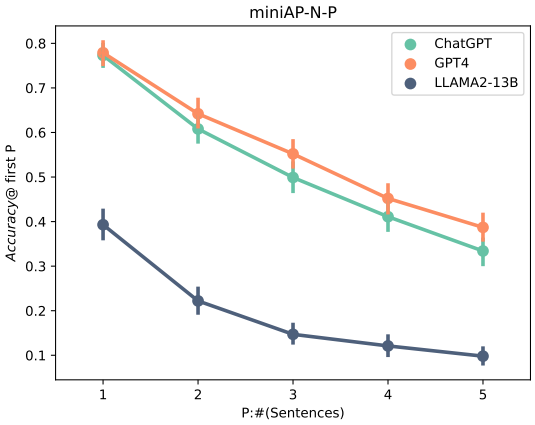


Figure 3: **Effect of input length on performance of models.** Accuracy at the first P sentences on the miniAP-N-P task. Note that as input text length increases, performance drops and all models display very similar trends.

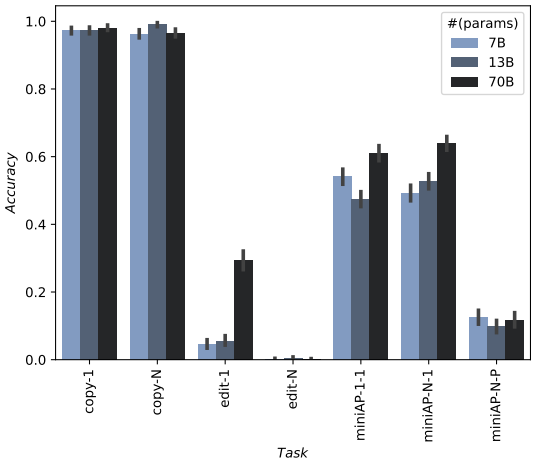


Figure 4: **Effect of model size on performance.** Performance is stable or increases with model size except on **Edit-N**.

sources, including the Internet, one might conjecture that LLMs have internalized many of the **miniAP**-style instructions. Thus we ask: *To what extent is model performance a reflection of instructions being well-aligned with prior knowledge internalized by models during their training phase?*. To shed some light on this, we measure model performance on counterfactual instructions that likely deviate from the LLM prior. The `miniAP-reversed` task consists of precisely these counterfactual instructions. Table 1 shows the result of this evaluation. First, we observe that there is a significant drop in the performance of ChatGPT (0.82 vs 0.61) and LLAMA2 (0.47 vs 0.25) in the counterfactual setting when compared to the likely well-aligned `miniAP` setting. This suggests that there is a significant effect of priors on model performance, and these models are hard to override. GPT4, on the other hand, shows some drop but not a significant one (0.71 vs 0.68), indicating that GPT4 is likely more steerable than the others. These findings are in line with Li et al. (2023); Wu et al. (2023), who make similar observations regarding the effect of prior knowledge on task performance.

Model	**Accuracy**	
	MiniAP (Natural)	Reversed MiniAP (Counterfactual)
CHATGPT	0.82	**0.61**
LLAMA2-7B	0.54	**0.43**
LLAMA2-13B	0.47	**0.25**
LLAMA2-70B	0.61	**0.47**
GPT4	0.71	0.68

Table 1: Performance of models (especially ChatGPT and LLAMA2) degrade when instructions are potentially not well-aligned with model priors.

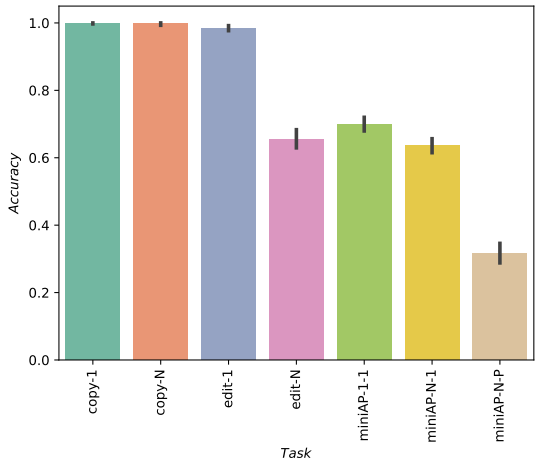


Figure 5: Performance when using a mixture-of-experts approach. Note the significant improvements on the **Edit** tasks.

Dimension	**Error %**	
	ChatGPT	LLAMA2
YEAR ABBREV.	7.6	7.3
TITLE ABBREV.	4.8	6.3
PUNCTUATION	15.2	21.6
PLURAL POSS.	24.2	7.5
NUMBER FORMATTING	23.6	21.3
COLON	18.5	17.0
CAPITALIZATION	6.2	19.0

Table 2: Error buckets of models on the miniAP task. Both model families struggle with instructions related to number formatting and punctuation *inter alia*.

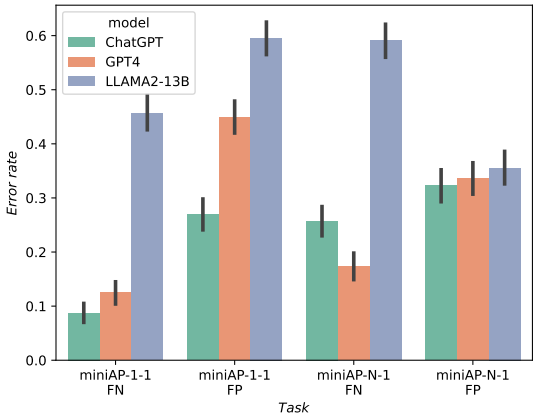


Figure 6: **Performance of models on miniAP-1-1 and miniAP-N-1 tasks broken down by two major error classes** (a) *False Negatives (FN)*: Error rate on records with non-compliant input (b) *False Positives (FP)*: Error rate on records with already compliant input text with no edits needed.

miniAP Task Error Analysis Error analysis of models on the **miniAP** task. There are two main error categories: **False Negatives** and **False Positives**. False negatives occur when the model either misses an error or incorrectly edits non-compliant. False positives occur when the model erroneously makes edits to fully compliant text.

Figure 6 shows the error rates for each of these classes for all models in the **miniAP-*-1** settings[4]. We find significant differences between models. First, LLAMA2 performs worse than the OpenAI models. Second, comparing ChatGPT and GPT4 in the **miniAP-1-1** setting, GPT4 tends to have a much higher false-positive rate, indicating that it is more aggressive than ChatGPT at making edits. However, in the multiple instruction setting (**miniAP-N-1**), GPT-4 has significantly lower false negatives than ChatGPT, indicating it is better at both detecting errors and rewriting the text to be compliant when given multiple instructions.

[4]We focus on single-sentence settings since it is easier to analyze errors on a single sentence and are representative.

What linguistic dimensions pose the most challenge? Table 2 shows the breakdown of the errors on the `miniAP-1-1` task for representative models from both families. Observe that all models generally find instructions related to ABBREVIATIONS easier than others. Instructions related to PLURAL POSSESSIVES, NUMBER FORMATTING, and COLON USAGE tend to be challenging for all models. This is likely because these instructions tend to be nuanced and have many exceptions.

Does Instruction Application via Code Generation Help? Many of the instructions in our tasks can actually be implemented as simple rules. It is, therefore, indeed sobering that LLMs may still find such instructions challenging to follow. That said, we ask whether we can leverage an adjacent ability of LLMs – namely, their ability to generate code to translate such instructions to computer code instead. This may offer a practical alternative with a few advantages: (a) **Improved consistency**–it might be easier to generate correct code for an instruction relative to directly following it (for e.g., **edit-N** instructions). (b) **Improved interpretability**: The generated code offers better interpretability and potential guarantees of correctness. (c) **Reduced error rates** – the resulting code can be applied consistently regardless of input size. (d) Obviously, while such instructions could be translated to code by human programmers, using LLMs can be much more scalable. We thus adopt a mixture-of-experts approach: First, we ask the LLM to identify instructions that can be implemented as simple Python functions and then generate corresponding code. The application of the full instruction set is as follows: (1) Apply the subset of instructions encoded as functions and (2) apply the remaining instructions by defaulting to the prompting strategy.

Figure 5 shows the results of this approach using GPT4 [5]. The results are mixed. Note the large gains in performance on the **Edit-1** (0.99 vs 0.5) and **Edit-N** (0.6 vs 0.2) tasks relative to our default prompting approach (compare to GPT4 results in Figure 2). This is because GPT4, by-in-large, generates correct code implementing these operations. However, on the **miniAP** tasks, the overall performance is slightly worse because it generates incorrect code for some of the instructions (e.g. number formatting). To conclude, such an approach can be effective when LLMs are able to translate human-readable instructions to correct code with a high enough precision to overcome the error rate of the default prompting strategy.

[5]We only consider GPT4 as it is known to be much better at code generation over ChatGPT and CodeLLAMA and thus establishes a ceiling over both.

5 Does Supervised Fine-tuning help?

We noted significant gaps in model performance in the zero-shot setting. Therefore, we ask: *Does further fine-tuning on potentially task-specific instruction-following data help boost performance?* Additionally, we ask *how well do such fine-tuned models generalize to unseen tasks like those we outlined in Section 2.2?* We investigate further supervised fine-tuning and consider task mixtures that progressively increase the number of held-out tasks:

- **all-main**: We use data from all the tasks outlined in Section 2.1.
- **1-Inst**: We only use data from single instruction settings from the tasks outlined in Section 2.1. This setting assesses whether models trained only on single instructions generalize to multiple instruction settings.
- **miniAP-held-out**: We use data from all tasks outlined in Section 2.1 except the `miniAP` task. Instead, we include GENERIC-TE, the large-scale generic text editing instruction dataset.
- **all-held-out**: We hold out all tasks and only train on GENERIC-TE.

For each main task included, we use $1,000$ training examples per task. We use all examples from GENERIC-TE when it is included.

5.1 Fine-tuning Procedure

We finetune the LLAMA2(13B) model using standard instruction-based fine-tuning with one minor difference: we use a "completions-only" loss where the loss is computed only on the expected completion (output tokens), although we attend to all tokens. This encourages the model to focus on learning to generate the required output rather than learning to auto-complete parts of the input. We train all models for 1 epoch on 1 A100 instance with 8 GPUs using a per-device batch size of 8.

5.2 Evaluation

Overall Performance. Tables 3 and 4 show the performance of instruction fine-tuned models. In every setting, the overall performance (see Table 4) improves significantly over the baseline. As expected, the gains on the seen tasks are the highest.

	copy -1	copy -N	edit -1	edit -N	miniAP -1-1	miniAP -N-1	miniAP -N-P	UL -1	UL -N	miniAP (Reversed)
Untrained	0.97	0.99	0.06	0.0	0.47	0.53	0.1	0.15	0.02	0.26
all-main	1.0	1.0	0.86	0.56	0.99	0.97	0.93	0.24	0.04	0.08
1-Inst	0.97	0.41	0.83	0.01	0.98	0.89	0.22	0.51	0.07	0.17
miniAP-heldout	1.0	1.0	0.93	0.58	0.86	0.74	0.42	0.55	0.1	0.58
all-heldout	0.93	0.44	0.13	0.01	0.85	0.65	0.46	0.1	0.0	0.53

Table 3: **Performance of supervised fine-tuned models.** Blue cells indicate those tasks are *held-out* in that setting.

	Average Accuracy		
	Overall	**Seen**	**Unseen**
Untrained	0.36	–	0.36
all-main	0.67	0.9	0.12
1-inst	0.51	0.93	0.33
miniAP-heldout	0.68	0.88	0.54
all-heldout	0.41	–	0.41

Table 4: Average performance of LLAMA models on tasks specific to different settings in supervised fine-tuning.

However, when trained on all the seen tasks, significant gains on seen tasks may result in a performance drop on unseen tasks, especially tasks that are counterfactual to the seen task (see **miniAP-reversed** for **all-main**:0.08 and **1-Inst**:0.17 compared to **Untrained**:0.26 in Table 3). Instruction fine-tuning on a small set of tasks adapts the model more aggressively to those tasks, thus limiting generalization to new tasks. There is, however, some evidence that single-instruction fine-tuned models can generalize to apply multiple instructions and outperform the baseline overall (0.51 vs 0.36 in Table 4). Finally, settings that include the GENERICTE dataset significantly boost overall performance, demonstrating value in training on a large and very diverse set of instructions (see **miniAP-heldout** and **all-heldout**).

6 Related Work

While there is a large body of work on instruction fine-tuning (Mishra et al., 2022; Iyer et al., 2022; Chung et al., 2022; Taori et al., 2023), explicitly evaluating the instruction-following capability of LLMs is relatively scarce with the most related work being Webson and Pavlick (2022); Kung et al. (2023); Zhou et al. (2023); Zeng et al. (2023). Webson and Pavlick (2022) investigate the extent to which prompt-based models understand their prompts and that note good model predictions even with misleading prompts, thus cautioning against attributing model performance to prompt understanding. Similarly, Kung et al. (2023) argue that many instruction fine-tuned models may be relying on prompt artifacts (eg., output format) and advise caution in ascribing superior performance to their instruction-following ability. Zeng et al. (2023) look at the related problem of evaluating the efficacy of "LLM-evaluators" – where an LLM and a prompting strategy are used to rank other LLM model outputs according to some specific criterion (e.g. sensitivity). They propose a benchmark called for evaluating such evaluators and note many evaluators vary in their preferred outputs showing significant room for improvement. Li et al. (2023) investigate the reliance of models on prior knowledge, encoded during training when following instructions (e.g., instructions on classifying text). They note that model performance deteriorates significantly when instructions deviate from the model's prior knowledge – a finding that is also supported by our work.

We differ from the above work in that we evaluate LLMs on their ability to follow fine-grained text-editing instructions across various dimensions in a controlled manner. We introduce a benchmark suite of tasks consisting of elementary text-editing instructions as well as text-editing instructions derived from popular style guides. Finally, we investigate the effect of various model parameters that can guide modeling improvements and practical applications.

7 Conclusion

We comprehensively investigated the instruction-following ability of LLMs in the context of text editing. We introduced specific tasks and benchmarks to enable a controlled evaluation and provide important insights into the effect of instruction parameters (number, complexity, and dependencies) as well as model parameters (scale, training methods) on model performance. Our findings can help inform LLM modeling efforts and application development.

Limitations

Our work has the following limitations. First, we focus only on English. It is quite possible that

the performance might be significantly different in other languages. Second, we focused on a subset of instructions of the AP style guide for which it was relatively easy to gather gold-standard compliant data and evaluate automatically. Consequently, we do not consider style guidelines that are subjective and/or related to pragmatic intent (For example: "Keep the audience in mind and make sure to encourage them to respond", "Make the text witty and engaging to the audience"). Third, our evaluation is limited to paragraph sized text. We do not characterize performance on very long inputs (like entire documents or entire style guides spanning hundreds of pages). While we expect performance to be worse for longer inputs, it is additionally possible that we may uncover new instructions that may prove to be challenging at the document level. Fourth, our characterization of OpenAI model performance is subject to limitations around their closed nature (e.g. model behavior might change over time and under our feet). Finally, it is important to recognize that we seek to characterize model performance favoring lower efforts in prompt engineering. We concede that with extensive prompt engineering, one may find prompts that improve model performance on our task, but then again, we argue that in such cases, such a model is less steerable when adjusted for the prompt engineering effort.

Ethical Considerations

We use text that is either computationally generated (edited) or generated via LLMs (in line with their terms of service). Our work does not introduce additional ethical considerations other than those broadly associated with large language models.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. 2022. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*.

Po-Nien Kung and Nanyun Peng. 2023. Do models really learn to follow instructions? an empirical study of instruction tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1317–1328, Toronto, Canada. Association for Computational Linguistics.

Po-Nien Kung et al. 2023. Do models really learn to follow instructions? an empirical study of instruction tuning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1317–1328, Toronto, Canada. Association for Computational Linguistics.

Shiyang Li, Jun Yan, Hai Wang, Zheng Tang, Xiang Ren, Vijay Srinivasan, and Hongxia Jin. 2023. Instruction-following evaluation through verbalizer manipulation. *arXiv preprint arXiv:2307.10558*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*.

R Thomas McCoy, Shunyu Yao, Dan Friedman, Matthew Hardy, and Thomas L Griffiths. 2023. Embers of autoregression: Understanding large language models through the problem they are trained to solve. *arXiv preprint arXiv:2309.13638*.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, Dublin, Ireland. Association for Computational Linguistics.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Nigar M Shafiq Surameery and Mohammed Y Shakor. 2023. Use chat gpt to solve programming bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290*, 3(01):17–22.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

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*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

Albert Webson and Ellie Pavlick. 2022. Do prompt-based models really understand the meaning of their prompts? In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2300–2344, Seattle, United States. Association for Computational Linguistics.

Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas, and Yoon Kim. 2023. Reasoning or reciting? exploring the capabilities and limitations of language models through counterfactual tasks. *arXiv preprint arXiv:2307.02477*.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating large language models at evaluating instruction following. *arXiv preprint arXiv:2310.07641*.

Jeffrey Zhou et al. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Data Availability

We will release the benchmark data publicly as a resource to the community for academic research purposes.

B Error analysis of Edit Task

Why do LLMs struggle to perform the Edit task. Based on our observations, we can make two conjectures: First, we note that even performance on the single instruction case is still only about 50% (GPT-4). This suggests that both ChatGPT and GPT4 find it challenging to follow basic operations on characters at specific indices consistently (note even with explicit instructions that 0-based indexing is used. We also noted that explicit CoT and implicit CoT prompting also did not yield significant improvement). Second, sequential dependency between instructions can be additionally challenging because models need to maintain an internal scratch buffer because the output at any step critically depends on the intermediate output at the previous instruction. Both these conjectures suggest that the final predicted output is likely to be close in "edit-distance" similarity to the expected output, and we might expect to see the accuracy and "edit-distance" similarity drop as the number of instructions increases. Figure 7 shows the accuracy and a measure of edit-distance-based similarity[6] on the test set stratified by a number of instructions. Indeed, we note that while the accuracy is low as expected, the edit-distance similarity is higher (closer to 0.6) and suggests that many characters in the predicted output match the characters from the expected output with a few errors.

C Effect of number of training examples over performance.

Figure 8 shows the performance of LLAMA models as a function of number of training examples per seen task in the supervised fine-tuned setting. Note that for simple tasks like the **Copy** task, only 10 examples are sufficient to guide the model to learn the specific task. In settings with multiple instructions or instructions with sequential dependencies, one might need up to 1000 examples.

[6]We use the measure $\frac{2M}{T}$, which will be between 0 and 1. M is the total number of character matches, and T is the total of characters in both prediction and the expected strings.

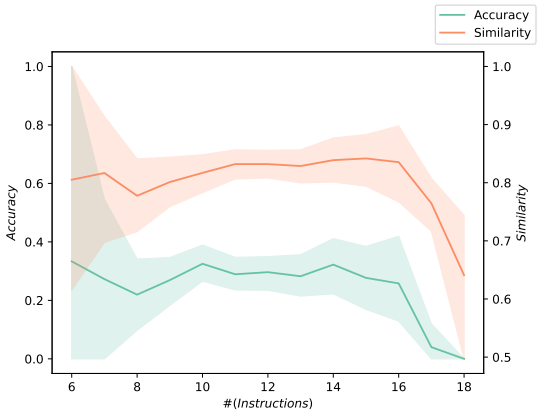


Figure 7: **edit-N task: Performance versus #(instructions).** Note that performance drops further when number of instructions is greater than 15.

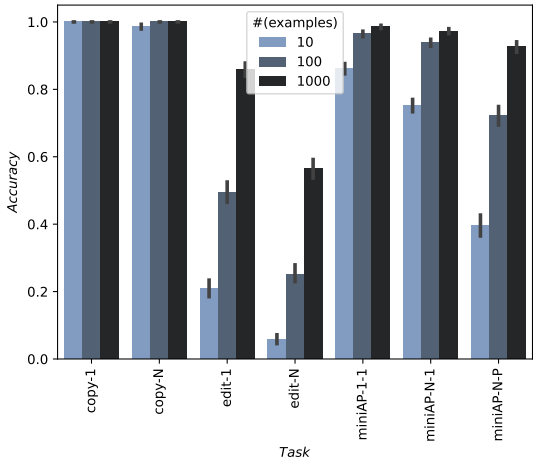


Figure 8: Performance of LLAMA models with super-vised fine-tuning on the benchmark tasks.

D Effect of Text-Editing instruction tuning on general task performance

Here, we document the effect of further text-editing instruction tuning on general task performance. We do this by evaluating the performance of our fine-tuned model on the following popular benchmarks: HELLASWAG, ARC_EASY, LAMBADA_OPENAI, MMLU and compare against the corresponding baseline model. Table 5 shows the result of this evaluation. We note that while there is a small drop in overall performance on these general benchmarks, as expected; our substantial improvement in the text-editing instructions more than compensates for this. Furthermore, this observation suggests that the slight drop in general NLP task performance in lieu of a significant improvement in a specific set of tasks might be a viable trade-off for the usage of such LLMs in practical settings, especially in text-editing applications.

	HELLA SWAG		ARC_EASY		LAMBADA_OPENAI	MMLU
	Acc (std.err)	**Acc Norm (std.err)**	**Acc (std.err)**	**Acc Norm (std.err)**	**Perplexity (std.err)**	**Acc (std.err)**
LLAMA13B	0.607(0.004)	0.796(0.004)	0.775(0.008)	0.737(0.009)	2.971(0.070)	0.531(0.138)
LLAMA13B (miniAP-heldout)	0.581(0.004)	0.777(0.004)	0.767(0.008)	0.720(0.009)	3.153(0.067)	0.498(0.135)

Table 5: Performance on general tasks of our text-editing instruction fine-tuned model against the corresponding baseline. Note that while there is some drop in performance compared to the baseline this is more than compensated by the very significant performance gain on text editing instructions, suggesting the tradeoff might be a viable one in practice where one might want to use LLMs for text-editing applications.

E miniAP Instructions

Figure 9 is the list of instructions in the miniAP task.

```
<styleguide>
<punctuation>
<instruction> Add an apostrophe if using a possessive form of a plural form ending in s.
</instruction>
<instruction> Add an apostrophe s if using a possessive form of a plural form not ending in s.
</instruction>
<instruction> Capitalize the first word after a colon if the word that follows a colon is a proper
noun. </instruction>
<instruction> Capitalize the first word after a colon if what follows a colon is a complete
sentence.
</instruction>
<instruction> Do not capitalize the first word after a colon if what follows is a list or a phrase
that does not begin with a proper noun. </instruction>
<instruction> Do not use [] but replace them with (). </instruction>
<instruction> Use slash without spaces to signify alternatives and not a hyphen. </instruction>
</punctuation>

<abbreviation>
<instruction> Abbreviate titles of persons before their name. </instruction>
<instruction> Abbreviate titles of persons (Junior, Senior) after their name. </instruction>
</abbreviation>

<dateandtime>
<instruction> Abbreviate years using two digits. When abbreviating a year with an apostrophe, the
apostrophe should be turned away from the date.
</instruction>
</dateandtime>

<capitalization>
<instruction> Capitalize only proper nouns and product names. </instruction>
<instruction> Do not capitalize season names except when they are the first word of a sentence.
</instruction>
</capitalization>

<numbers>
<instruction> For single-digit numbers, write them in words.  For eg.  9 apples => Nine apples.
</instruction>
<instruction> Write numbers with more than 5 digits using groups of three digits each starting
from the right end. If there is a decimal point, the grouping should only be on the integer side
of the decimal point. </instruction>
<instruction> Use . as a decimal separator when using the decimal point. </instruction>
</numbers>
</styleguide>
```

Figure 9: List of instructions in the miniAP task.