Expanding Batch Prompting to Zero-Shot and Multi-Task Regimes

Alex Chandler
The University of Texas at Austin
alex.chandler@utexas.edu

Rohan Jha
The University of Texas at Austin
rjha@utexas.edu

Abstract

Recent works leveraging the in-context learning capabilities of modern large language models (LLMs) have independently proposed variations of a novel technique: Batch Prompting (BP), wherein b question prompts are indexed and concatenated into a single BatchPrompt. The queried LLM then learns from batch-formatted in-context examples how to answer all b questions in one generation. In this work, we consider sufficient conditions of a BatchPrompt in order to explore the technique's efficacy in the zero-shot and multi-task scenarios. We present new BatchPrompt templates and demonstrate on both open- and closed-source models of varying size that BP is possible without few-shot exemplars and also robust to in-batch task diversity provided sufficient instructions. We conclude with a modification to the token efficiency metric η proposed by the original work and a discussion of which regimes of NLG are best suited to the technique.

1 Introduction

Many tasks in NLP, including question answering, sentiment analysis, and named entity recognition, which once required specialized training techniques and intricate pipelines to achieve SOTA results, are now rivaled by a general LLM with a prompt of well-chosen instructions and in-context exemplars (Nori et al., 2023). These prompts, however, contribute to the large cost (measured in both compute and time, by way of tokens) of LLM inference. Batch Prompting (Cheng et al., 2023) is an inference technique that concatenates a batch of b questions into one prompt submitted for a multi-answer response. It reduces a job's inference cost by amortizing the cost of shared instruction and few-shot example tokens over b samples. We expand upon the specific mechanism of BP's inference efficiency in Section 5.

One limitation of current BP methods is the model's dependence on learning multi-answer formatting from a fixed $k = b$ few-shot in-context examples of the general form `Q[i], A[i]`. Unfortunately, many domains and use cases remain where it is not feasible to provide annotated examples at inference time. To address this, we present in Section 3.1 the techniques to build a BatchPrompt that instead draws on the instruction components laid out by Lin et al. (2023) in an entirely zero-shot manner. Experimental results presented in Section 4.4 indicate that zero-shot batch prompting is possible across multiple batch sizes (b). However, when compared to the few-shot setting, there is a notable increase in the rate of misformatted outputs for zero-shot BatchPrompts.

Encouraged by the promising results of zero-shot BP, we further examine the potential negative effects of mixing diverse tasks within a batch which we hypothesized as a consequence of prior work on distracting context (Shi et al., 2023). Surprisingly, results presented in Section 4.5 demonstrate BP's robustness to multi-task mixing within a batch.

Our contributions are: 1) We introduce two new BatchPrompt templates: the first designed for single-task reasoning and the second for multi-task reasoning. Both are effective in zero-shot and few-shot settings, providing unambiguous meta-instructions for answer formatting and task-specific instructions for adaptability. 2) We demonstrate the capability of our new BatchPrompt to work with k in-context exemplars across multiple batch sizes b including the zero-shot $k = 0$ setting, albeit at the cost of increased formatting errors. 3) In multi-task batch scenarios, we present evidence that increasing the diversity of tasks within a BatchPrompt while controlling for batch size does not result in any noticeable decline in the performance of individual tasks. 4) We extend Cheng et al. (2023)'s explanation of BP's token efficiency η, and identify

how other LLM techniques impact BP's efficiency.

2 Related Works

Batch Prompting, initially proposed by Cheng et al. (2023), involves a straightforward process of concatenating k selected in-context examples with b queries, indexed as `Q[i]` and `A[i]`, into a single prompt, enabling a language model to answer all questions in a single inference call to an LLM. They validate both its effectiveness and efficiency (with token cost scaling theoretically inversely linear to b) on batch sizes up to $b = 6$ on a range of common sense QA, arithmetic reasoning, and NLU/NLI benchmarks. Cheng et al. found that 1) the particular scheme of selecting in-context examples did not show improvement over random and 2) BP performance suffers mild degradation with batch size, which steepens with task complexity. Approximately contemporaneously[1] with the original work, Lin et al. (2023) similarly develops a more verbose BatchPrompt that incorporates instructions for intermediate reasoning techniques such as chain-of-thought (Wei et al., 2023). Furthermore, they scale their experiments to a much larger batch size ($b = 64$) and subsequently propose an iterative voting strategy over permuted intra-batch orders to mitigate the greater observed performance degradation at larger batch sizes. This large-batch regime, however, has only recently been made feasible by advancements in attention mechanisms and long-context training that permit, for example,[2] the 32k-token context window of GPT-4 (OpenAI, 2023).

3 Methodology

3.1 Engineering and Designing Robust Prompt Batching Templates

We use OpenAI Playground and GPT-4 (OpenAI, 2023) to prototype prompt templates that consistently formatted output across various tasks. These templates were laboriously engineered to ensure both consistent and reliable output, enabling the creation of one universal cross-task answer parsing function that can extract batched answers as a list. We observe that minor modifications to the prompt can substantially increase the batch parsing error rate, and that even slight ambiguities in the prompt contribute to a higher rate of LLM parsing errors. Our templates share a common structure, with task-specific descriptions replacing generic placeholders. Our prompt templates incorporate several key features to ensure robustness:

- **Comprehensive Task Description**: The single-task BatchPrompt template starts with sections labeled "Objective", an optional "Complexity", and a "Method" section. These sections collectively offer a well-rounded understanding of the task. The "Objective" clarifies the primary goal, "Complexity" provides an estimate of the task's intricacy, and "Method" outlines the recommended approach for task execution. The multi-task BatchPrompt template includes an instruction, method, intermediate reasoning, and output meaning section for each task ensuring that the LLM has sufficient instruction to answer each task in the same BatchPrompt. When needed, the prompts contain extra instructions defining the output meaning, clearly mapping the LLM's response to an integer. This integer can denote output classes for each question in the batch, facilitating easier parsing.

- **Structured Instructions**: Both single-task and multi-task BatchPrompt templates feature a concise numbered list of instructions; the instructions direct the model to explicitly output the steps used to obtain each answer, specify the precise number of questions to answer, and instruct the model to answer all questions, even in cases of uncertainty.

- **Input and Output Format Specification**: The single-task and multi-task BatchPrompt templates rigorously define formats for input questions and output answers to ensure clarity and facilitate parsing. The input format for both single-task and multi-task BP templates employs indexed questions, denoted as `Q[i]`.

  For the multi-task BatchPrompt, each question includes a `task_code`, formatted roughly as `Q[i][task_code]: {question prompt}`, aiding in associating each query with the correct task description in the BatchPrompt. The single-task BatchPrompt follows a similar indexed format without the `task_code`.

[1] Lin et al. (2023) was published on ArXiv 8 months after Cheng et al. (2023) with a footnote acknowledging its late discovery of the original work.

[2] We assume, based on current research, as GPT-4 remains shrouded in secrecy.

Following (Lin et al., 2023), the phrase `The answer is` is mandated to precede each answer to maintain uniformity in the output, which in turn aids in simplifying the parsing and evaluation of responses. As a result, the output format for both templates is structured as `A[i]: {intermediate_reasoning}; The answer is {answer}`.

These features collectively contribute to the robustness and reliability of our prompt templates, making them useful tools for generating consistent and specified output in batch processing scenarios. We share our complete Batch Prompt templates in the appendix.

4 Experiments

4.1 Datasets

We assess the effectiveness of Batch Prompting across four datasets similar to those assessed in Cheng et al. (2023), measuring the accuracy of each task across our experiments. Our evaluations cover a range of task-oriented datasets: mathematical reasoning, represented by GSM8K (Cobbe et al., 2021); common sense reasoning with CommonsenseQA (Talmor et al., 2019); and natural language inference (NLI), through the MNLI and RTE datasets under the GLUE benchmark (Wang et al., 2019).

4.2 Metrics

We evaluate the individual performance of each tasks in our batch using accuracy across all experiments. Additionally, we monitor **Batch Parsing Error Rate**, which quantifies the collective frequency of two types of parsing failures: 1) when an answer is not generated for a given question, and 2) when the generated answer does not conform to the format specified by the prompt.

4.3 Experimental Setups

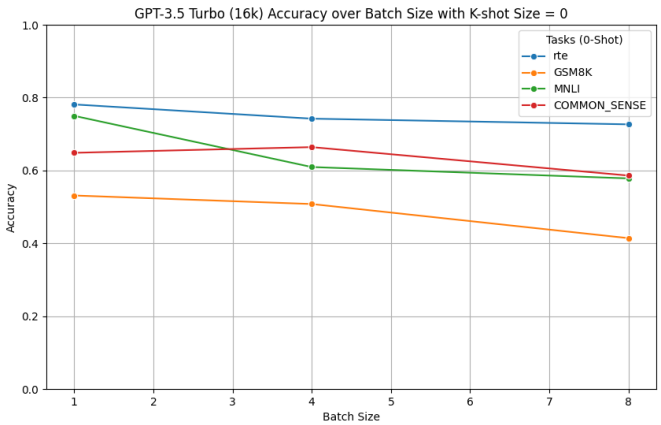
We conduct our experiments using three large language models. We use GPT-3.5-16K-Turbo for our experiments due to its larger context window allowing for larger batches and more in-context examples than its GPT-3 predecessors (Brown et al., 2020). We also test on the 13B and 70B versions of the open Llama-2 model (Touvron et al., 2023). In each experiment, we evaluate the initial 240 samples from each dataset to stay within budget constraints and minimize carbon emissions. Our API costs total to $70.46. We fix the temperature parameter to 0 for all three models during inference to enhance experiment replicability and as a sensible setting for un-creative short answer tasks. Preliminary experiments conducted with a temperature of 0.3, show no increase in batch parsing error rate, nor other experimental trends particularly counter to those presented in this paper. We use OpenAI's commercial API platform (ope) to query GPT-3.5-16K-Turbo, and TogetherAI's commercial API platform (tog, 2023) API to query full-bit-precision versions of Llama-2-13B and Llama-2-70B models.

4.4 Single-Task Batch Prompting Results

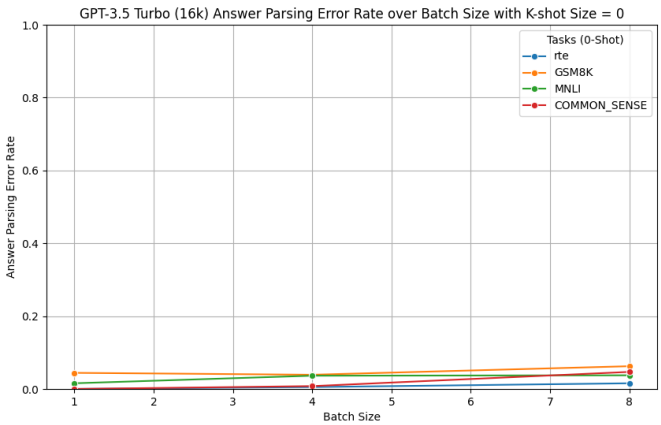
Model	k	b	Accuracy (%)
Llama-2-13B	0	1	36
		4	43
	1	1	48
		4	32
Llama-2-70B	0	1	55
		4	54
	1	1	59
		4	57
GPT-3.5-16K-Turbo	0	1	68
		4	63
		8	58
	1	1	72
		4	69
		8	62
	6	1	75
		4	69
		8	68

Table 1: Average accuracy (computed over RTE, MNLI, GSM8K, and CSQA) of models in zero-shot and few-shot settings (k) with increasing prompt batch sizes (b).

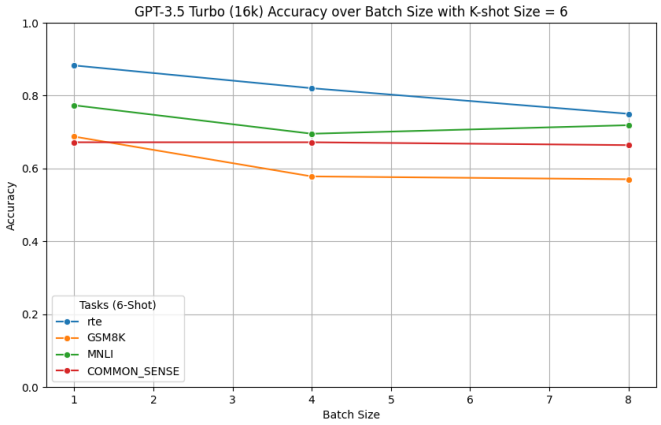
Our analysis of single-task performance across RTE, GSM8K, MNLI, and Commonsense QA shows distinct trends. Figure 1a shows the zero-shot accuracy of each task in a zero-shot setting. 1c shows the same accuracy of each task in a few-shot setting with $k = 6$. In the both charts, a decrease in task performance is evident as Batch Size increases. Conversely, figures 1b and 1d both show a rise in batch parsing error rate with larger batch sizes. Similar patterns emerge in tests on Llama-2-70B, detailed in appendix A.2. However, Llama-2-13B presents a contrasting pattern, showing improved



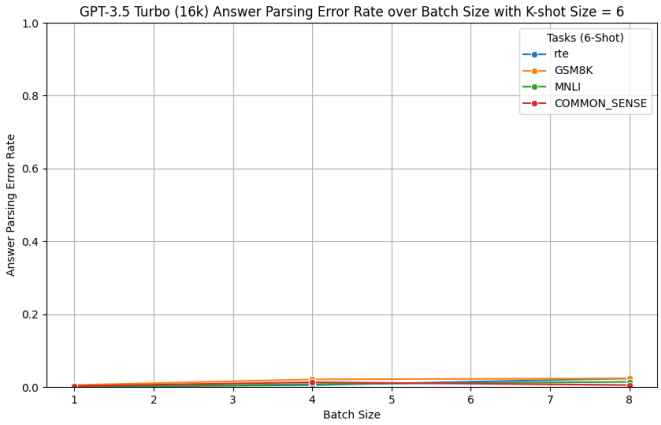
(a) Zero-Shot Accuracy



(b) Zero-Shot Batch Parsing Error Rate



(c) $k = 6$-shot Accuracy



(d) $k = 6$-Shot Batch Parsing Error Rate

Figure 1: Single-task, Few- and Zero-Shot Batch Prompting Accuracy and Batch Parsing vs Batch Size.

accuracy with larger batches in zero-shot settings as seen in Figure 12 in the appendix. Given the overall low accuracy in Llama-2-13B when $b = 1$, we infer that the diminished performance at lower batch sizes may be attributed to Llama-2-13B's inability to effectively understand and process our BatchPrompts in zero-shot scenarios; we thus focus our analysis on experimental trends from GPT-3.5-16K-Turbo and Llama-2-70B.

4.5 Multi-Task Batch Prompting Results

We next test the performance of our zero-shot multi-task BatchPrompt on all 15 combinations of the 4 benchmark datasets. Each experiment evaluates 240 samples of each task mixed in batches of 4 such that each batch has at least one sample from the given combination of tasks. We record in table 2 the accuracy and batch parsing error rates of each task in each combination. Figure 2 plots these measures against number of tasks. We observe no significant change in performance with respect to number of tasks in a batch prompt, which we interpret as BP being robust to multi-task prompts when provided with a sufficient task description for *each task*.

Table 2 and figure 3 also show that there is no clear benefit nor detriment to having similar (e.g. RTE and MNLI – similar NLI tasks from GLUE) or different (e.g. RTE and GSM8K – NLI vs arithmetic reasoning) tasks within a batch. This is counter to our original hypothesis that more, and especially very different, tasks in a batch would negatively affect performance following from work on distracting contexts. Both Llama-2 models' accuracies and batch parsing error rates are plotted in appendix A.2.

Batch ($b = 4$)	Task Accuracy (%)				Task Parse Error (%)			
	CSQA	GSM8K	RTE	MNLI	CSQA	GSM8K	RTE	MNLI
C	67	-	-	-	4	-	-	-
G	-	47	-	-	-	11	-	-
R	-	-	65	-	-	-	3	-
M	-	-	-	60	-	-	-	6
C,G	**69**	48	-	-	4	**4**	-	-
C,R	61	-	72	-	6	-	4	-
C,M	66	-	-	**64**	6	-	-	**3**
G,R	-	47	70	-	-	7	7	-
G,M	-	42	-	56	-	11	-	5
R,M	-	-	65	60	-	-	5	4
C,G,R	66	46	70	-	6	8	**3**	-
C,G,M	62	41	-	53	**3**	7	-	3
C,R,M	60	-	**73**	55	7	-	3	3
G,R,M	-	42	68	63	-	6	4	4
C,G,R,M	65	**55**	71	58	6	5	4	4

Table 2: Per-task Accuracy and Batch Parsing Error Rates for Multi-Task batch combinations (b=4) on GPT-3.5-16K-Turbo. Tasks in a batch are labeled by their first letter: (C)SQA, (G)SM8K, (M)NLI, (R)TE. 240 samples per dataset per combination. Highest accuracy/lowest error is **bolded** for each task.

5 Discussion

5.1 Batch Prompting Efficiency

5.1.1 Token Efficiency

In their work, (Cheng et al., 2023) define the token efficiency for standard prompting as $\eta_{standard} = \frac{1}{K+1}$ and for batch prompting as $\eta_{batch} = \frac{b}{K+b}$, where K denotes the number of in-context exemplars, and b is the number of samples in a batch. However, these definitions assume that the in-context token count K is identical for both standard and batch prompting ($K_{batch} = K_{standard}$), an assumption that fails to capture the practical complexities of batch prompting. In reality, batch prompts typically require additional tokens to prime the model to answer all questions in the desired output format. Considering these factors, we propose a refined formulation specifically for token efficiency in batch and standard prompting. Let K_{batch} and $K_{standard}$ represent the total input tokens (including in-context examples, instructions, and query tokens) for batch and standard prompts, respectively, with $K_{batch} > K_{standard}$. Let K_{batch} represent the total number of input tokens for batch prompts, and $K_{standard}$ for standard prompts, excluding the query tokens. This includes all in-context examples and instructions. Note that K_{batch} is greater than $K_{standard}$. In the multi-task setting, the discrepancy between K_{batch} and is $K_{standard}$ even larger, as seen in BatchPrompt examples in appendix A. Then, let Q_{batch} represent the average number of input and output tokens required per batched query, and $Q_{standard}$ for a standard query, to both ask and answer the question. $Q_{batch} > Q_{standard}$ to account for the indexing tokens $Q[i]$ for questions and $A[i]$ for answers. The revised token efficiencies can be defined as:

$$\eta_{standard} = \frac{1}{K_{standard} + Q_{standard}} \quad (1)$$

$$\eta_{batch} = \frac{b}{K_{batch} + Q_{batch} b} \quad (2)$$

To determine the point at which batch prompting surpasses standard prompting in terms of input token efficiency, we solve for the condition $\eta_{batch} > \eta_{standard}$. After rearranging and solving for b, the condition for batch prompting to have higher token efficiency than standard prompting is found to be:

$$b > \frac{K_{batch}}{K_{standard} + Q_{standard} - Q_{batch}} \quad (3)$$

This inequality presents a more realistic and stricter bound on the efficiency of batch prompting and the necessary size of b to justify the use of BP for natural language tasks. It accounts for the additional tokens required in batch prompting for managing multiple queries and specifying the expected input and output formats.

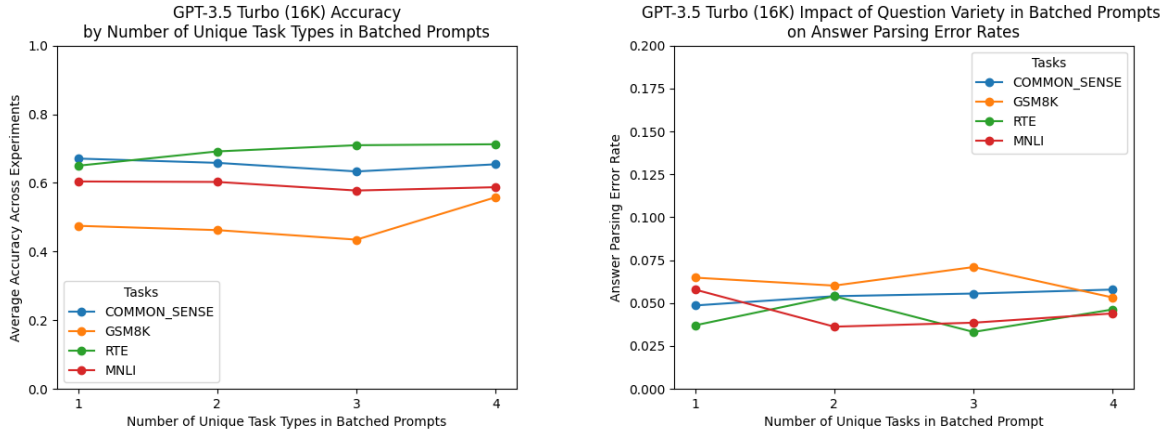


Figure 2: Task Accuracies and Batch Parsing Error Rates vs batch task diversity (# tasks in batch) tested on GPT-3.5-16K-Turbo. All batches are size $b = 4$ and task metrics are averaged over all combinations of the same size. Similar plots for the Llama-2 family of models can be found in the appendix.

5.1.2 Time Efficiency

We define time efficiency for prompting as the ratio of questions answered to the time taken. For standard prompting, it's represented by $\frac{1}{T_{encode\_standard}+T_{decode\_standard}+T_{api}}$, with $T_{encode\_standard}$ and $T_{decode\_standard}$ as the input encoding and output decoding times, respectively, and T_{api} as the API network wait time overhead. For batch prompting, time efficiency can be measured as $\frac{b}{T_{encode\_batch}+T_{decode\_batch}+T_{api}}$, where T_{encode_batch} and T_{decode_batch} are the respective encoding and decoding times for batch queries, and b is the batch size. Notably, T_{encode_batch}, T_{decode_batch}, $T_{encode\_standard}$, and $T_{decode\_standard}$ could change as model architectures evolve. In contrast to our detailed token efficiency section, we have chosen not to further dissect the relationship between input encoding and output decoding token times to batch size in the time efficiency analysis. This decision stems from the presence of numerous confounding factors that complicate deriving a meaningful relationship, particularly for the majority of LLM users who depend on API-based interactions. Given T_{api} and API Token limits as the current primary constraints to time efficiency, batch prompting could practically provide up to b-times more time-efficiency over standard prompting. An alternative to BP that would enhance time efficiency is independently passing numerous prompts each through separate API calls. However, considering token rate limits as a potential bottleneck for time efficiency, our discussion and formulas on token efficiency may provide more insights to comparing the time efficiency of batch prompting with standard prompting. We defer the empirical analysis of quantifying the wall clock benefits of BP as a subject for future research.

5.2 Limitations and Future Work

Despite its demonstrated capability to operate in zero-shot and multi-task settings, BP still faces key limitations. First and most clearly, the existence of parsing errors (particularly in zero-shot) that don't exist with single prompts is undesirable. These errors are persistent and demand a lot of prompt engineering to be suitably addressed. We, however, suspect that this issue could be alleviated using either answer format validation with retries (Liu, 2023) or a constrained decoding strategy using an automatically-generated grammar based on batch size and task descriptions (Scholak et al., 2021; Geng et al., 2023). We leave this hypothesis to future works.

In addition, Batch Prompting's token efficiency improvement is limited to splitting the cost of *shared input* tokens. This means that while BP saves tokens when a prompt is comprised of many in-context examples and shared instructions (such as those in our zero-shot BatchPrompt), it cannot save on sample-specific tokens of each task such as question prompts or generated response tokens. Consequently, Batch Prompting may fail to significantly enhance efficiency gains on expensive reasoning techniques like chain- or tree-of-thought (Wei et al., 2023; Yao et al., 2023) while consuming the model's context window at b times the rate. These efficiency gain limitations point towards BP being most useful in settings with easy, short out-

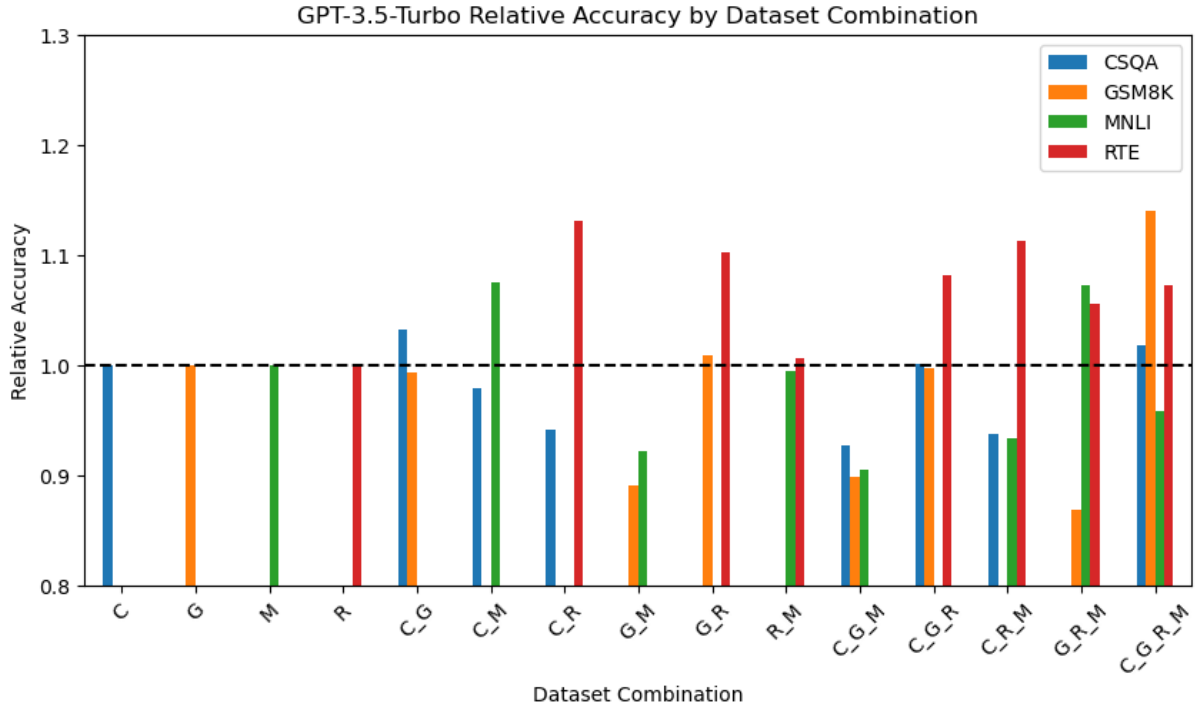


Figure 3: Relative (to single-task case) task accuracies for each task combination using GPT-3.5-16K-Turbo and $b = 4$.

puts (such as with the datasets we test) that require minimal reasoning to achieve sufficient accuracy. We believe that future work on BP should be directed in these domains that are best suited to its strengths such as a dynamic batch sizing technique and associated experiments quantifying the relation between performance decline and task complexity beyond using length as a proxy. Future research should investigate fine-tuning a Large Language Model to minimize the number of input tokens required for efficient Zero-Shot Single-Task and Multi-Task Batch Prompting, while also aiming to lower the Answer Parsing Error Rate.

6 Conclusion

Batch Prompting has been demonstrated as a simple means of reducing an LLM workload's token cost at a minor cost to accuracy and formatting errors that might be mitigated by ensemble voting and constrained decoding strategies. We present a further examination of BP in the zero-shot domain by developing an example-free set of instructions used to construct a BatchPrompt. Experiments on both GPT-3.5-Turbo-16K and Llama-2-70B demonstrate that the loss of in-context examples is not catastrophic to performance with the correct format instructions. We also expand this prompt to the multi-task batch setting and demonstrate the invariance of overall performance to batch task diversity.

Our zero-shot single-task and multi-task BatchPrompts lead to a broader understanding of BP's token efficiency η and its limitations. It highlights the specific contexts where BP is most beneficial and points towards potential areas for future research.

7 Project Contributions

We have split the workload of this project fairly and in a way that preserves autonomy of tasks. Rohan implemented the generic experiment framework that constructs the batch prompts and queries the models in frequent design conversation with Alex. Alex implemented evaluation code and has conducted the model queries and prompt tuning to arrive at a suitably general prompt for all cases. Throughout the writing and implementation stages, we've kept each other well-informed regarding specific implementation details to allow for different experiment configurations (datasets, k-shot, model, etc), and what metrics and configurations to support/aim for. We also share the responsibility of devising the given set of experimental configurations to execute to test our research questions, as

well as writing proposals and reports.

References

Openai api. https://openai.com/api/.

2023. Togetherai. https://www.together.ai/.

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

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. Batch prompting: Efficient inference with large language model apis.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured nlp tasks without finetuning.

Jianzhe Lin, Maurice Diesendruck, Liang Du, and Robin Abraham. 2023. Batchprompt: Accomplish more with less.

Jason Liu. 2023. Better data extraction using pydantic and openai function calls. Published on July 26, 2023.

Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, Renqian Luo, Scott Mayer McKinney, Robert Osazuwa Ness, Hoifung Poon, Tao Qin, Naoto Usuyama, Chris White, and Eric Horvitz. 2023. Can generalist foundation models outcompete special-purpose tuning? case study in medicine.

OpenAI. 2023. Gpt-4 technical report.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901. Association for Computational Linguistics.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge.

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, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models.

A Appendix

A.1 Example Single-Task BatchPrompt: GSM8K Zero-Shot

Batch Prompt Example

Objective:
Your task is to solve a set of math questions in a batch. The total number of questions in the batch is defined as {batch_size}.
Complexity:

Each question in the batch will require you to perform between 2 and 8 steps to arrive at the final answer.
Method:
Use basic arithmetic operations to perform a sequence of calculations for solving these questions.

Instructions:

1. **Intermediate Reasoning:** Include all the intermediate steps you took to arrive at each answer. This ensures that the answer is both well-reasoned and reliable.

2. **Batch Size:** The number of answers you provide must exactly match the specified {batch_size}.

Input Format:

```
Q[0]: {Question_0_Text}
Q[1]: {Question_1_Text}
...
Q[{batch_size - 1}]:
{Question_{batch_size-1}_Text}
```

Output Format:

```
A[index]: {Intermediate_
Reasoning}; The answer is
{Answer_Integer}
```

- index: This is the index of the question you are answering. It must be prefixed with 'A' and enclosed in square brackets.

- {Intermediate_Reasoning}: This is where you provide all the intermediate steps that led you to the final answer.

- {Answer_Integer}: This is the final integer answer to the question.

The phrase 'The answer is' must directly precede the integer answer and come after the intermediate reasoning, separated by a semicolon. Please adhere strictly to these guidelines to ensure the output is in the desired format.

A.2 Multi-Task BatchPrompt - Various Domains

Batch Prompt Example

Objective:
Your task is to solve a variety of questions across multiple domains in a single batch operation. The total number of questions in the batch to answer is defined as batch_size = {{batch_size}}.

Task Descriptions:

COMMON_SENSE: Instruction - our task is to solve a set of multiple-choice questions from the CommonsenseQA dataset in a batch. CommonsenseQA is a new multiple-choice question answering dataset that requires different types of commonsense knowledge to predict the correct answers . You will be given {{batch_size}} questions each time, as input. These questions are designed to test your ability to answer queries that often require contextual understanding and general world knowledge. Your goal is to select the letter corresponding to the most appropriate answer among five options labeled 'a', 'b', 'c', 'd', and 'e' for each question in the batch. COMMON_SENSE: Method - Use your expertise in NLP and contextual understanding to perform a sequence of logical evaluations for solving these questions. COMMON_SENSE: Intermediate Reasoning - Include all the steps you took to arrive at your answer. This could include identifying key phrases, contradictions, or logical connections that led you to choose a particular option. COMMON_SENSE: Output Meaning - Select the most appropriate answer and output the letter after "The answer is" with the corresponding letter.
GSM8K: GSM8K: Instruction - Your task is to solve a set of math questions in a batch. GSM8K: Method - Use basic arithmetic operations to perform a sequence of calculations for solving these questions. GSM8K: Intermediate Reasoning - Each question in the batch will require you to perform between 2 and 8 steps to arrive at the final answer. GSM8K: Output Meaning - Each answer is an integer that is the answer to the

question.

RTE: RTE: Instruction - Your task is to solve a set of recognizing textual entailment (RTE) questions in a batch. You will be given {{batch_size}} sentence pairs from the Textual Entailment Recognition dataset each time, as input. Your goal is to classify each sentence pair into two classes. RTE: Method - Use your expertise in NLP and sentence pair relationship annotation to perform a sequence of logical evaluations for solving these questions. RTE: Intermediate Reasoning - Include all the steps you took to evaluate the relationship between the Premise and Hypothesis. This could include identifying key phrases, contradictions, or logical connections. RTE: Output Meaning - An answer of 0 signifies entailment between the Hypothesis and Premise, while 1 signifies non-entailment.

MNLI: MNLI: Instruction - Your task is to solve a set of MultiNLI (MNLI) questions in a batch. You will be given premise-hypothesis pairs from the MNLI dataset as input. Your goal is to classify each pair into one of three classes: entailment, neutral, or contradiction. MNLI: Method - Use your expertise in NLP and sentence pair relationship annotation to perform a sequence of logical evaluations relationship between each Premise and Hypothesis pair. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral). MNLI: Intermediate Reasoning - Include all the steps you took to evaluate the relationship between the Premise and Hypothesis. This could include identifying key phrases, contradictions, or logical connections. MNLI: Output Meaning - An answer of 0 means the premise entails the hypothesis, indicating that if the premise is true, the hypothesis must also be true. In this case, the information in the hypothesis is a logical subset of the information in the premise. An answer of 1 means the relationship between the premise and the hypothesis is neutral, suggesting that the truth of the premise neither guarantees nor contradicts the truth of the hypothesis. The hypothesis could be either true or false regardless of the premise's truth value. An answer of 2 means the premise contradicts the hypothesis, implying that both cannot be true at the same time. If the premise is true, the hypothesis must necessarily be false, and vice versa.

Instructions:

1. **Intermediate Reasoning:** Include all the steps you took to evaluate the relationship between the Premise and Hypothesis.
2. **Batch Size:** You must provide an answer for each question in the batch, ensuring that the number of answers you provide exactly matches the specified {{batch_size}}.
3. **Handling Ambiguities:** Answer every question even if you are unsure about the answer.

Unified Input Format:

The batch will contain questions each tagged with a unique index and a task code, formatted as follows:
Q[index][task_code]: {{Question_Text}}

Output Format:

You must adhere to the following format rigorously for each answer:
A[index]: {{Intermediate_Reasoning}};
The answer is {{Answer_Integer}}

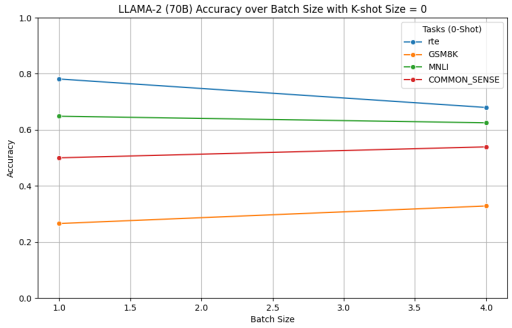


Figure 4: Line Plot illustrating the Accuracy of Zero-Shot Single-Task Batched Prompts with Llama2-70B at various batch sizes

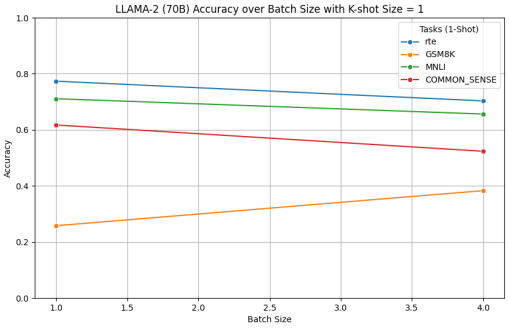


Figure 6: Line Plot illustrating the Accuracy of One-Shot Single-Task Batched Prompts with Llama2-70B at various batch sizes

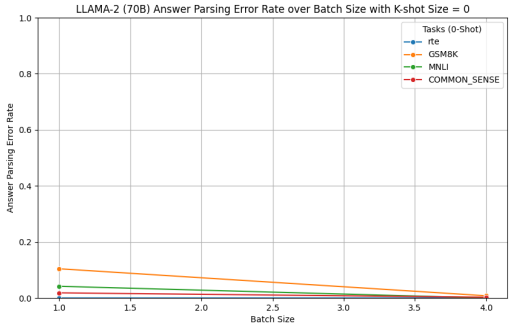


Figure 5: Line Plot illustrating the Batch Parsing Error Rate of Zero-Shot Single-Task Batched Prompts with Llama2-70B at various batch sizes

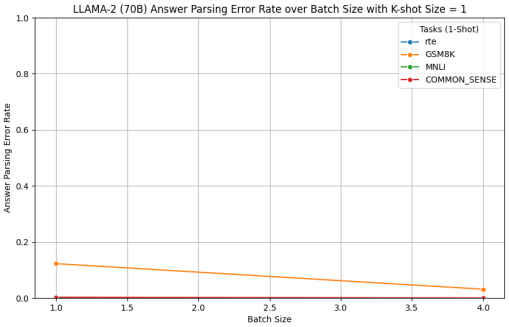


Figure 7: Line Plot illustrating the Batch Parsing Error Rate of One-Shot Single-Task Batched Prompts with Llama2-70B at various batch sizes

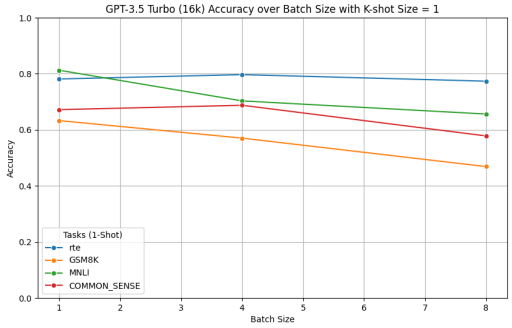


Figure 8: Line Plot illustrating the Accuracy of One-Shot Single-Task Batched Prompts with GPT-3.5-Turbo-16K at various batch sizes

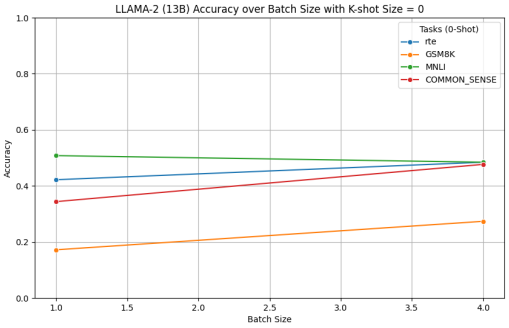


Figure 10: Line Plot illustrating the Accuracy of Zero-Shot Single-Task Batched Prompts with Llama2-13B at various batch sizes

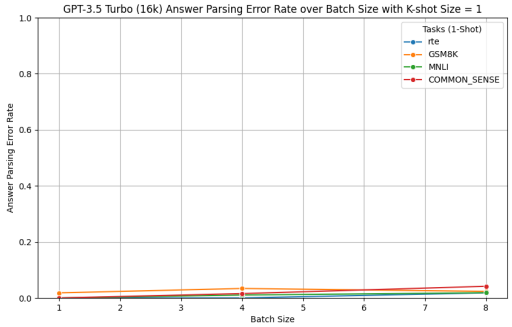


Figure 9: Line Plot illustrating the Batch Parsing Error Rate of One-Shot Single-Task Batched Prompts with GPT-3.5-Turbo-16K at various batch sizes

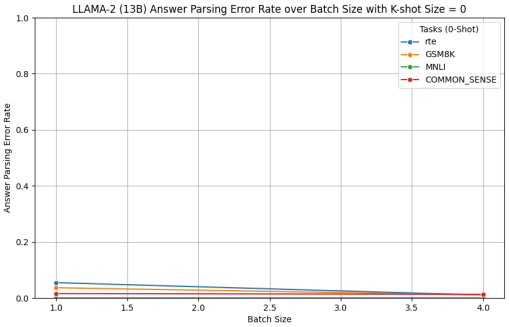


Figure 11: Line Plot illustrating the Batch Parsing Error Rate of Zero-Shot Single-Task Batched Prompts with Llama2-13B Turbo at various batch sizes

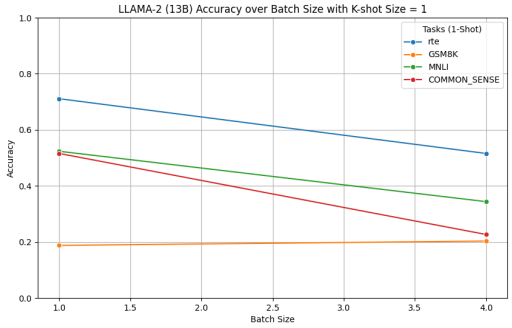


Figure 12: Line Plot illustrating the Accuracy of One-Shot Single-Task Batched Prompts with Llama2-13B at various batch sizes

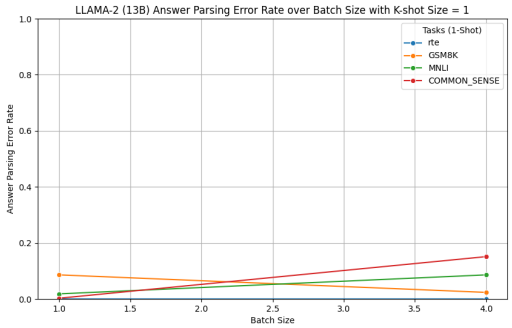


Figure 13: Line Plot illustrating the Batch Parsing Error Rate of One-Shot Single-Task Batched Prompts with Llama2-13B Turbo at various batch sizes

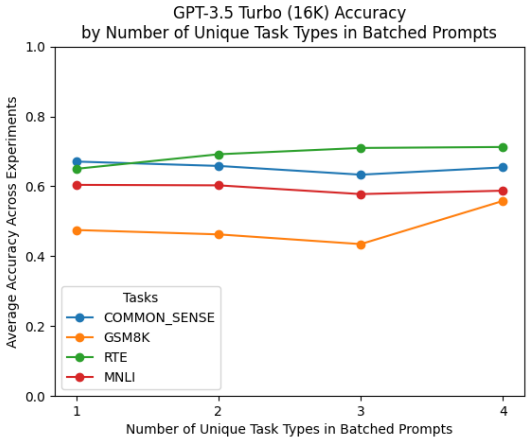


Figure 14: Line Plot illustrating the Zero-Shot Accuracy of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for GPT-3.5-Turbo-16K

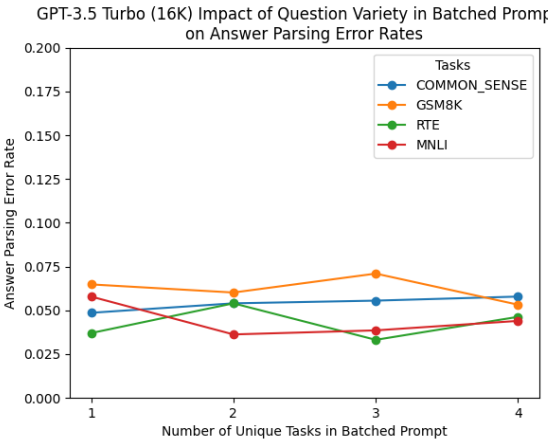


Figure 15: Line Plot illustrating the Batch Parsing Error Rate of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for GPT-3.5-Turbo-16K

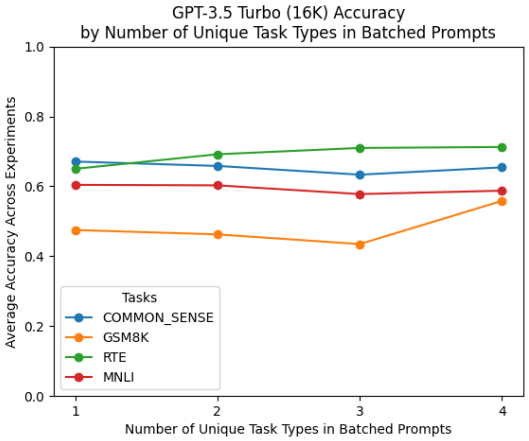


Figure 16: Line Plot illustrating the Zero-Shot Accuracy of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for GPT-3.5-Turbo-16K

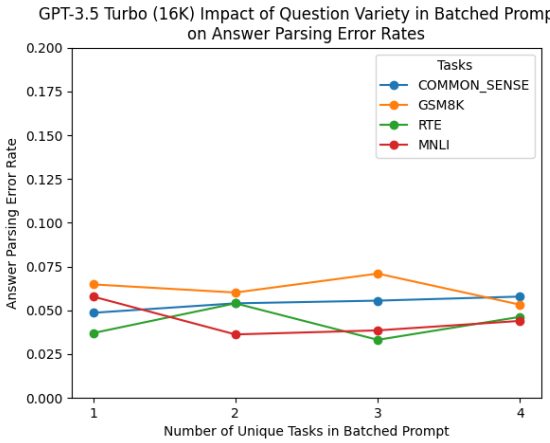


Figure 17: Line Plot illustrating the Batch Parsing Error Rate of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for GPT-3.5-Turbo-16K

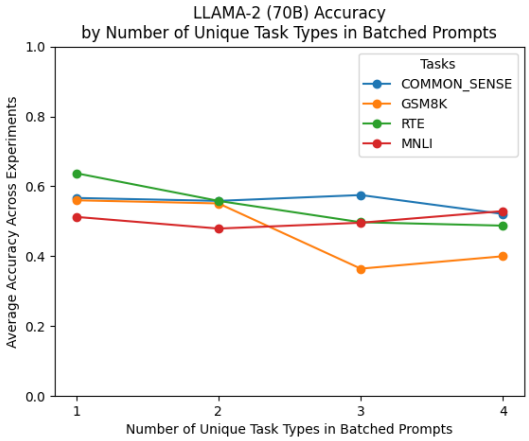


Figure 18: Line Plot illustrating the Zero-Shot Accuracy of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for Llama2-70B

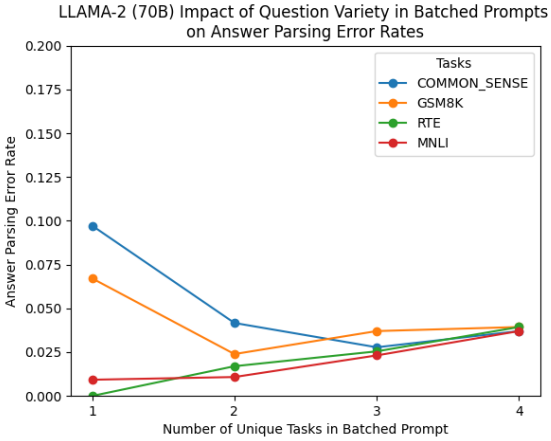


Figure 19: Line Plot illustrating the Batch Parsing Error Rate of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for Llama2-70B

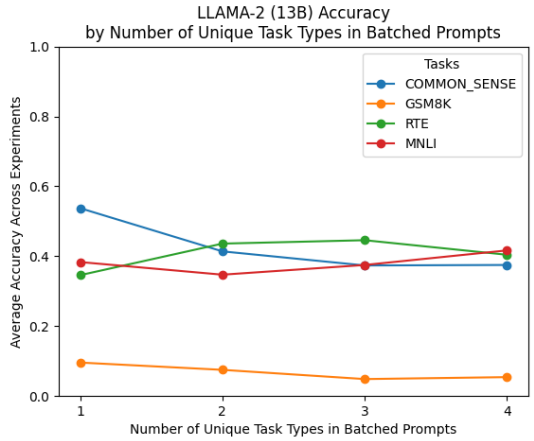


Figure 20: Line Plot illustrating the Zero-Shot Accuracy of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for Llama2-13B

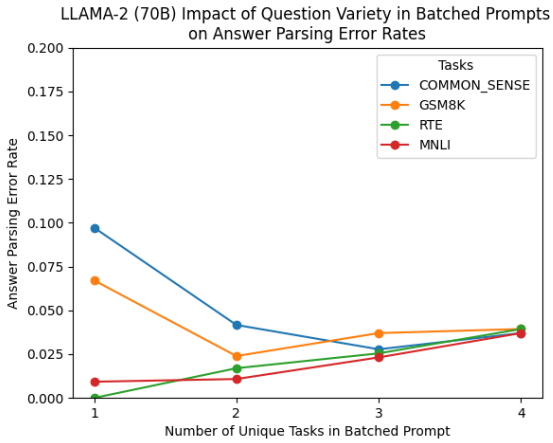


Figure 21: Line Plot illustrating the Batch Parsing Error Rate of Each Task against the Number of Unique Task Types in Multi-Task Batched Prompt for Llama2-70B