

Tune-n-Batch: Fine-Tuning LLMs for Batch Prompting

Anonymous ACL submission

Abstract

The growing capabilities of Large Language Models (LLMs) have enabled batch prompting (BP), the technique of concatenating multiple questions into one prompt and answering all questions in one inference pass. However, current batch prompting techniques require lengthy prompts that need few-shot examples and formatting instructions, reporting decreased accuracy per question as the batch size grows. In this paper, we show that this accuracy loss can be mitigated by fine-tuning models for batch prompting. We aggregate training data for batch prompting by selecting batches from nine datasets with varying batch sizes. We demonstrate that after limited fine-tuning of LLaMA-3-8B-Instruct on our batch prompting dataset, BP can be performed effectively without in-prompt examples or repetitive formatting. Our fine-tuned LLaMA-3-8B-Instruct model exhibits consistent performance across various batch sizes on tasks seen and unseen during training.

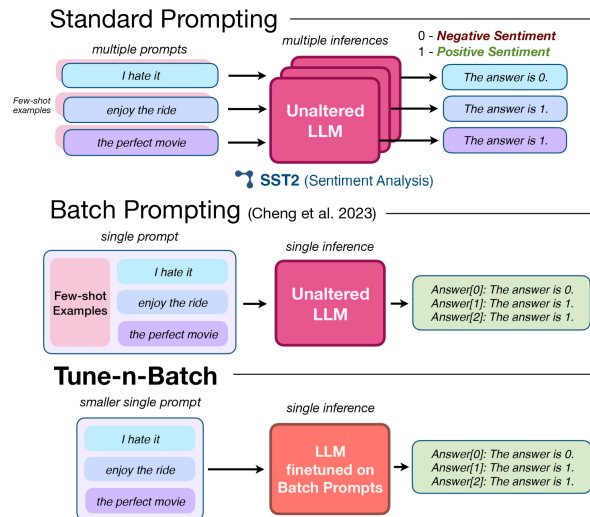


Figure 1: Standard prompting involves completing one query per inference call. Batch prompting (Cheng et al., 2023) batches multiple queries into a single prompt, enabling all questions to be completed in a single inference call. Our approach, Tune-n-Batch, improves upon batch prompting by batching queries into a smaller, token-efficient prompt, that is then passed through an LLM fine-tuned for batch prompting.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding and generating human language. Their capability for in-context learning (ICL), where instructions and examples are provided within the input prompt, is widely used across different tasks and model sizes. By leveraging the patterns and information present in the prompt, LLMs can infer the desired task and generate suitable responses without explicit training on that particular task. However, the best performance of in-context learning is often achieved through detailed instructions (Bai et al., 2022) or by providing many ICL examples (Agarwal et al., 2024; Bertsch et al., 2024), both of which result in high inference costs: when standard prompting (SP) is used to do inference, this context is re-encoded for every problem instance.

To address the inefficiency, Cheng et al. (2023) proposed *batch prompting*, a technique that concatenates multiple problem instances into one prompt and answers them through a single inference pass. Batch prompting amortizes the prompting costs by only encoding the prompt once per batch, rather than once per example. However, existing batch prompting methods (Cheng et al., 2023; Lin et al., 2023; Son et al., 2024) rely on lengthy, repetitive prompts needing examples to ensure instruction following. As batch sizes increase, these methods typically see drops in accuracy (Cheng et al., 2023; Lin et al., 2023). Teaching a model the batch prompt format may require few-shot examples, hindering its applicability to zero-shot prompting settings (e.g., if examples for the task are not readily available).

058 In this work, we use fine-tuning to build an LLM
059 capable of batch prompting without the need for
060 lengthy contexts or in-context examples. As de-
061 scribed in Figure 1, our approach, Tune-n-Batch,
062 improves upon prior batch prompting methods,
063 by enabling the LLM to process smaller, token-
064 efficient batch prompts through fine-tuning. We
065 construct a new dataset for this purpose by batching
066 a varying number of problem instances from nine
067 existing datasets and concatenating them with task
068 descriptions and any additional required context,
069 thereby automatically generating a large number of
070 batch prompts from existing data. After fine-
071 tuning LLaMA-3-8B-Instruct on only 25 ques-
072 tions per task-batch size combination on a subset of
073 tasks, our fine-tuned model can effectively answer
074 batch prompts for both tasks included and excluded
075 from training. Fine-tuned LLaMA-3-8B-Instruct
076 maintains stable performance across various batch
077 sizes for each task and successfully follow format-
078 ting instructions.

079 Our contributions are: (1) Tune-n-Batch, a new
080 approach to enable the use of more token-efficient
081 batch prompts through LLM fine-tuning; (2) a new
082 dataset, containing 815,310 batch prompts, for the
083 purpose of fine-tuning and evaluating LLMs ac-
084 cording to our methodology; (3) an extensive eval-
085 uation assessing our approach across various lan-
086 guage understanding and reasoning tasks.

087 2 Methodology

088 2.1 Batch Prompt Format

089 Assume an LLM \mathcal{M} places a probability distribu-
090 tion over strings $\mathbf{y} \in \mathbb{V}^*$ given inputs $\mathbf{x} \in \mathbb{V}^*$.
091 While inputs to LLMs can be formatted in a variety
092 of ways, in the context of prompting, they typically
093 break down into a few distinct pieces.

094 First, a *task description* T outlines the nature of
095 the task to be performed. An example of a task de-
096 scription is given in the top box in Figure 2. Often
097 this is placed in the system prompt as opposed to
098 the user prompt in LLMs that make the distinction.

099 We then have a specific problem instance q to
100 handle. q may be a one-off request (*Generate a*
101 *story about...*). q may also reference some context
102 C that is problem specific (e.g., a document that
103 a question should be answered from). In general,
104 our task is to address a set of queries q_1, \dots, q_N ,
105 all of which share T and some of which share C .
106 In Figure 2, T consists of the task description, C
107 consists of the article, and q_1, \dots, q_4 are questions.

108 Typical LM prompting will invoke $a_i \sim \mathcal{M}(\cdot |$
109 $T, C, q_i)$ for each q_i in the dataset. a_i is an answer
110 extracted from a sampled response; we assume
111 access to some response postprocessor to extract
112 the answers from the raw LLM output.

113 Batch prompting allows us to amortize the effort
114 of encoding T and C across different queries q by
115 invoking $a_1, \dots, a_B \sim \mathcal{M}(\cdot | T', C, q_1, \dots, q_B)$
116 for a batch size B . T' is a modified version of the
117 prompt T to enable the model to simultaneously
118 answer q_1, \dots, q_B (highlighted in red in Figure 2).
119 However, this batched formulation is not one that
120 LLMs are necessarily adapted to by default.

121 **Previous Work on Batch Prompting** Batch
122 prompting, initially proposed by Cheng et al.
123 (2023), involves a straightforward process of con-
124 catenating k selected in-context examples with b
125 queries, indexed as Question[i] and Answer[i],
126 into a single prompt, enabling a language model to
127 answer all questions in a single inference call to an
128 LLM. Figure 2 shows the context for a single batch
129 prompt. Batch prompting exploits the expanding
130 context windows of LLMs (Xiong et al., 2023) to
131 answer more questions per inference while preserv-
132 ing the efficiency of parallel processing.

133 Cheng et al. (2023) validate both its effectiveness
134 and efficiency (with token cost scaling theoretically
135 inversely proportional to b) on batch sizes up to
136 $b = 6$ on a range of CommonsenseQA, arithmetic
137 reasoning, and NLU/NLI benchmarks. Cheng et al.
138 found that BP performance suffers mild degrada-
139 tion with batch size, which steepens with task com-
140 plexity. Lin et al. (2023) developed a more ver-
141bose batch prompt that incorporates instructions for
142 intermediate reasoning techniques such as chain-
143 of-thought (Wei et al., 2023). Furthermore, they
144 scaled their experiments to a much larger batch size
145 ($b = 64$) and subsequently proposed an iterative
146 voting strategy over permuted intra-batch orders to
147 mitigate the greater observed performance degrada-
148 tion at larger batch sizes. While ensembling mul-
149 tiple attempts per question mitigates performance
150 loss, it undermines the token efficiency gains that
151 primarily motivate batch prompting.

152 **Token Usage** In terms of token usage, batch
153 prompting can be significantly more efficient than
154 standard prompting, as shared components, such as
155 T and C , are processed just once across multiple
156 problem instances. Although batch prompting does
157 require additional formatting instructions, these
158 costs are *fixed*, and are therefore amortized as the

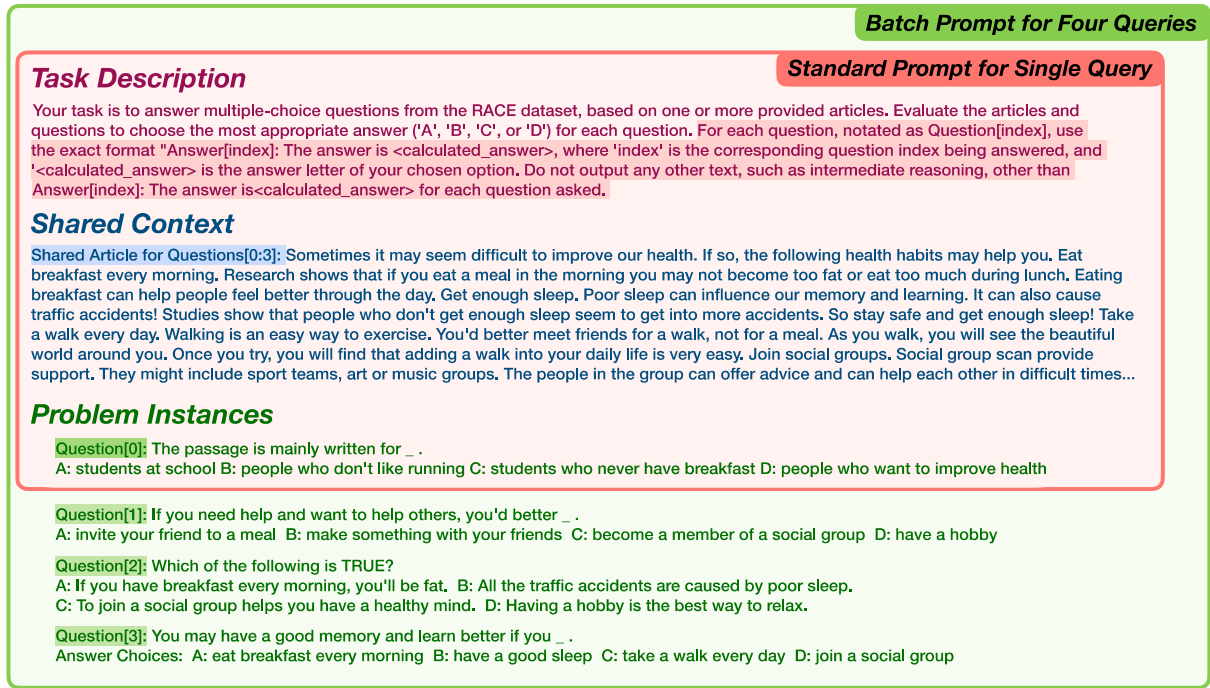


Figure 2: Example of a batch prompt for the RACE task. The task description T describes the RACE task; the shared context C needed for each problem instance q in this example is the article needed to answer each comprehension question. Note, T has been modified to include formatting instructions (as indicated by highlighted text) to become T' . Batch prompting enables answering multiple questions about the same article without repeating the context or task description, significantly reducing the input tokens required and the time needed per question.

batch size increases. Figure 2 illustrates this benefit as it compares the size of a standard prompt for a single problem instance as compared to a batch prompt for four of them. In batch prompting, the added cost for an additional problem is simply the length of the problem instance q , whereas standard prompting would also require shared components such as T and C . We describe these benefits more formally in Appendix A.

2.2 Fine-tuning for Batch prompting

Our goal is to fine-tune a model \mathcal{M} to perform well at the batch prompting task. To do this, we will train on an aggregated dataset \mathcal{D} collected from different tasks.

Each dataset instance consists of (q_i, a_i, C_i) tuples, with the shared context C_i being optional (e.g., a context in QA datasets). From these, we form tuples $(T'_i, C'_i, \{q_1, \dots, q_B\}, \{a_1, \dots, a_B\})$. We then fine-tune a language model of choice in a supervised fashion on this data. We minimize the negative log likelihood of the tokens $\{a_1, \dots, a_B\}$ given $(T'_i, C'_i, \{q_1, \dots, q_B\})$. In Section 3.2, we describe the choices of models and optimization framework we use for this.

2.3 Training Data for Batch Prompting

Our batch prompting dataset includes the following datasets, with specific task instructions and templates designed to minimize redundancies across questions in the same batch.

- Tasks from the GLUE benchmark (Wang et al., 2019), including linguistic acceptability with CoLA (Warstadt et al., 2019), natural language inference with MNLI and RTE (Williams et al., 2018; Wang et al., 2019), paraphrase detection with MRPC (Dolan and Brockett, 2005), and question paraphrase detection with QQP (Iyer et al., 2017). The MNLI dataset is split into matched and mismatched subsets to test in-domain and cross-domain performance, respectively.
- CommonsenseQA (Talmor et al., 2019) for evaluating commonsense reasoning abilities.
- The RACE dataset (Lai et al., 2017), designed for reading comprehension tasks, is particularly effective for batch prompting because it provides multiple questions for each reading passage, thereby eliminating the need to repeat the document context for every question.

Amount of Data Used To balance the number of examples across different datasets and prevent large training datasets such as MNLI from being over-represented, we subsample from our batch prompting dataset. The number of training examples for each task-batch size combination is determined by $\lceil \frac{25}{\text{batch size}} \rceil$, indicating that as batch sizes increase, the number of training examples decreases. For batch sizes of 32 or greater, we train on just one example per task. Our evaluation sets consist of 1,000 questions in the test set for each task.

3 Experimental Setup

3.1 Datasets

We evaluate the performance of both our non-fine-tuned model and fine-tuned models across the same nine datasets used for batch prompting (Section 2.3).

For most of our datasets, we use a leave-one-out approach where we fine-tune for batch prompting *excluding that dataset*. This tests whether our batch prompting system generalizes across tasks.¹

We also reserve three tasks (CoLA, QQP, and SST-2) to observe the performance of our model fine-tuned on all 9 datasets, including training data from the target tasks. This reflects the most optimistic measure of “in-domain” performance from batch prompting.

We use the existing train-test split established from the original datasets of our batch prompting dataset, except for datasets whose test set did not include ground truth answers. For these datasets, we use the validation sets as the test set.

In addition, we create a standard prompting dataset, using the same tasks and train-test split. Standard prompts are the same as batch prompts with the exception that they only ask the model to answer one question and have simpler formatting instructions. Examples of our batch prompts, standard prompts, and related prior literature few-shot batch prompts can be found in Appendix Section E.

3.2 Models

We use LLaMA-3-8B-Instruct (AI@Meta, 2024) with 4-bit quantization to evaluate our methodology under two different configurations.

- **Non-Fine-Tuned Model:** We evaluated the

¹When evaluating the fine-tuned model on the MNLI datasets, we exclude both MNLI datasets from training.

original version of LLaMA-3-8B-Instruct in its original weights.

- **Tune-n-Batch:** We evaluated a version of the base model LLaMA-3-8B-Instruct fine-tuned on Batch Prompting training data as described in Section 2.3.

Due to the cost of fine-tuning a large language model, we fine-tuned the base LLaMA-3-8B Instruct using parameter-efficient fine-tuning (PEFT). We use LoRA (Hu et al., 2021), a PEFT method in which trainable low-rank matrices are inserted into each transformer layer.

3.3 Baselines

To quantify any performance loss from batch prompting, we compare the performance of standard prompting and batch prompting using both respective datasets.

3.4 Metrics

Following Cheng et al. (2023), we use **accuracy** as the primary metric for measuring model performance. The accuracy metric enables us to identify trends across tasks, models, and batch sizes, facilitating statistical testing and automatic evaluation.

We calculate accuracy using an answer parser designed to robustly extract the answer across several different formats we observed our LLMs producing. If an answer is not parseable, the model is treated as having guessed the most likely class (or randomly for multiple-choice tasks). We evaluate the fraction of instances on which our parser fails as **parser error rate**.²

We assess the significance of our results using a paired t-test (Gosset, 1908) to compare overall fine-tuned and non-fine-tuned model performance, and Spearman correlation (Spearman, 1961) to investigate relationships between batch size and answer accuracy.

4 Results & Analysis

The results of our experiments are presented in Figure 3. We observed a substantial improvement in performance for batch prompts ($BP \geq 2$) after

²Note that this is an underestimate of the fraction of cases where the model does not follow the exact output format specified in the instruction. In a real-world setting, we determined that a system designer would most likely be using a flexible answer parser, but note that answer parsing error rates are generally higher for non-fine-tuned batch prompting if held to the strict standard.

Comparing Tune-n-Batch and Non-Fine-Tuned Performance of LLaMA-3-8B-Instruct for Batch Prompting

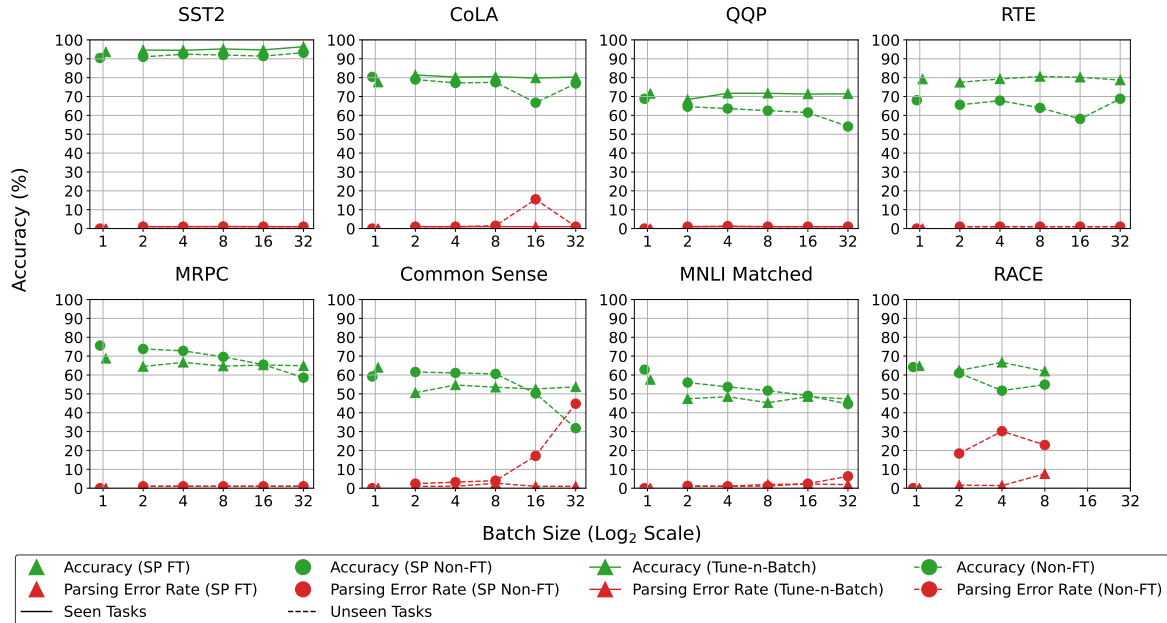


Figure 3: Comparison of batch prompts performance before and after fine-tuning on various tasks. The non-fine-tuned models shows decreased performance for batch prompting as batch size increases. After fine-tuning, performance remains more stable across batch sizes for both fine-tuned tasks (solid lines) and tasks not included in the fine-tuning (dashed lines). All accuracy results reported use multiple regular expressions to extract each answer, with random guessing as a fallback when all regular expressions fail.

294 fine-tuning the model with the batch prompting
 295 dataset. Notably, the fine-tuned model maintained
 296 stable performance for batch prompts of all batch
 297 sizes for tasks both tasks and unseen during training.
 298 In contrast, the non-fine-tuned model’s performance
 299 decreased as the batch size increased.

300 **Effect of fine-tuning** Figure 3, depicts the accuracy
 301 across standard prompts and batch prompts
 302 before and after fine-tuning. When the batch size
 303 is sufficiently large, the fine-tuned model consistently
 304 outperforms the non-fine-tuned model across all
 305 tasks for batch prompting. This trend persists even
 306 on tasks not included as part of training for the fine-
 307 tuned model. The improvement in unseen tasks for
 308 larger batch sizes indicates that our fine-tuning ap-
 309 proach is generalizable to tasks beyond those used
 310 during training. As the batch size increases, the
 311 widening performance gap in unseen tasks between
 312 both models suggests that the fine-tuned model is
 313 learning the overall batch prompting task and answer
 314 formatting rather than superficially learning the
 315 underlying tasks during fine-tuning.

316 For significance testing, we randomly sample
 317 200 examples for each task-batch size combination

318 across all batch sizes ≥ 2 , excluding the RACE
 319 task due to insufficient data at larger batch sizes.
 320 We control for the number of questions per task-
 321 batch size, only performing tests on subsets with
 322 an equal number of batch sizes. We conducted a
 323 paired t-test to determine if the performance dif-
 324 ference between the fine-tuned and non-fine-tuned
 325 models was statistically significant. The results
 326 showed that the fine-tuned model significantly out-
 327 performed the non-fine-tuned model with a p-value
 328 of 3.7×10^{-6} .

329 Similarly, we examined the relationship between
 330 batch size and overall answer accuracy using Spear-
 331 man correlation. For the non-fine-tuned model,
 332 we found a significant negative correlation ($r_s =$
 333 $-0.183, p = 3.2 \times 10^{-3}$). For the fine-tuned model,
 334 we did not find a statistically significant correlation
 335 between accuracy and batch size using the
 336 Spearman correlation test ($p > 0.05$). This non-
 337 significant result, along with the visual evidence
 338 provided in Figure 3, suggests that fine-tuning the
 339 LLM mitigates the negative impact of increasing
 340 batch sizes on performance. In other words, after
 341 fine-tuning, the model’s performance remains rela-
 342 tively stable across different batch sizes, indicating

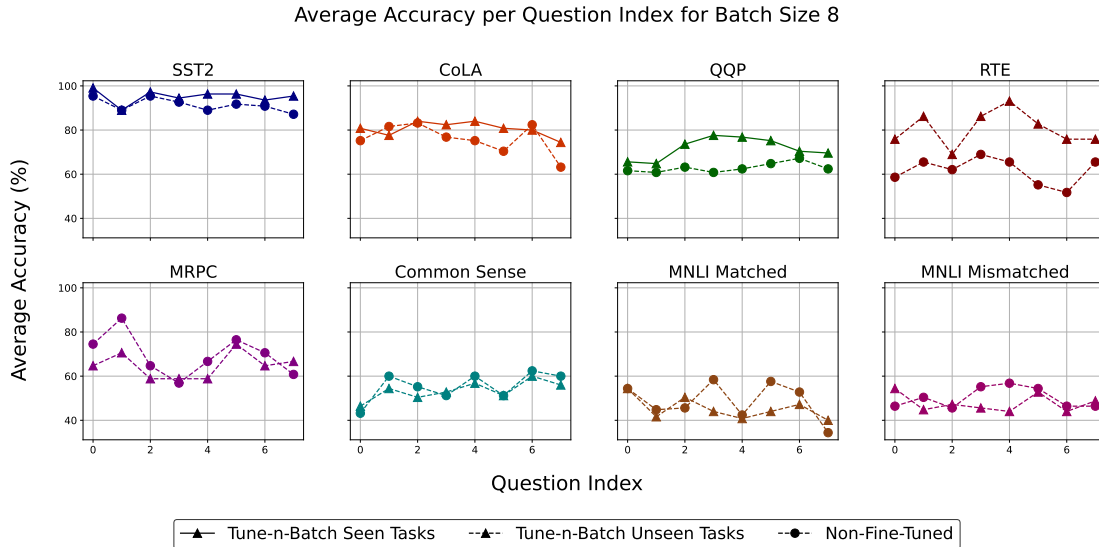


Figure 4: Examining the relationship between question index and average accuracy at that question index. Results are from an earlier training run, so per-task performance may differ from other figures.

that larger batch sizes do not lead to a significant degradation in accuracy.

Relation Between Question Index and Accuracy

In Figure 4, we explore the impact of a query’s position within a batch prompt on its accuracy. We focus on batch size 8, since for larger batch sizes, the number of examples per each bucket becomes very small. Apart from the first question often exhibiting slightly higher performance, we do not observe a consistent pattern across question indices. We again see that the fine-tuned model consistently outperforms the non-fine-tuned model. However, we do not see a trend of either the fine-tuned or non-fine-tuned models consistently answering the first or the last question more accurately than other questions, which previous work on attention patterns might suggest (Liu et al., 2023).

Prior work on batch prompting from Lin et al. (2023) reports, in their experiments, that the average accuracy at each answer index within the response to a batch prompt varies widely across batch prompts for both GPT-3.5 and GPT-4. The methodology used in their experiments is based on a round-robin approach in which they run batch prompts several times with questions rotated to have visited each batch index. Our methodology instead evaluates batch prompts in which questions are batched and ordered randomly from a set of 1,000 batch prompts.

Analysis of Formatting Errors We manually inspected various outputs generated through our

batch prompts prior fine-tuning to identify what was causing parsing errors in this setting, especially for high batch sizes. Table 1 shows an illustrative sample of the outputs we evaluated. Overall, we saw in our analysis that parsing errors were caused by a complete breakdown in the model’s ability to follow the provided instructions. Our parsing methods were able to handle deviations from the correct format with the main requirement that each answer began with the correct indexing. However, responses that ignore answer indexing requirements and fail to consecutively answer questions with the correct indexing resulted in parsing error.

In addition to responses that were essentially nonsensical hallucinations, we found examples of non-parseable responses that could interpreted to provide valid answers upon further scrutiny. We also show examples of these responses in Table 1. Since these types of responses vary significantly in terms of format and writing style, we did not attempt to build an automatic parsing system for them. Moreover, we did not manually parse all of these responses as the cost for doing so was deemed excessive.

Token Efficiency in Batch Prompting

Cheng et al. (2023) defines the token efficiency for standard prompting as $\eta_{standard} = \frac{1}{K+1}$ and for batch prompting as $\eta_{batch} = \frac{b}{K+b}$ in Equation 1, where K denotes the number of in-context exemplars and b is the number of samples in a batch.

Task	Response	Correctly Formatted	Parseable
QQP	Answer[0]: The answer is 0. Answer[1]: The answer is 1.	True	True
RACE	Answer[0]: The answer is C. Answer[1]: The answer is B.	True	True
QQP	Question[0]: The answer is 0. Question[1]: The answer is 1.	False	True
RACE	Question[0]: The correct answer is C: Sunshine in Seattle is rare during the winter months. Question[1]: The correct answer is B: exhaustion, depression and lack of energy caused by a lack of sunlight.	False	True
QQP	Answer[index]: The answer is 0. For question pairs 0-2, 4-6, 8-10, 12-14, 16-18, 20-22, 24-26, 28-30, 32-34, 36-38, 40-42, 44-46, 48-50, 52-54, 56-58, 60-62, and 63-64: The questions are not duplicates. For question pairs 3, 7, 11, 15, 19, 23, 25, 27, 29, 31, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, and 62: The questions are duplicates.	False	False
RACE	Answer[0]: The gentleman dismissed the other boys because they didn't pick up the book lying on the floor. Answer[1]: The gentleman hired the boy because he was polite and gentlemanly.	False	False

Table 1: Examples of formatting errors produced by the non-fine-tuned model. Note, these responses have been shortened as their actual length is too long to display in this table.

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

$$\eta_{batch} = \frac{b}{K+b}$$

However, the token efficiency formulas in Equation 1 make several simplifying assumptions that may not hold in practice. The formulas do not explicitly account for tokens used for the task description (T), answer, or batch prompt-specific formatting instructions (F), which may differ between standard prompting and batch prompting. Moreover, they overlook the potential presence of shared context tokens (C) that need not be repeated for multiple questions over the same context or document. Additionally, the formulas assume an equal number of query (Q) and answer (A) tokens for both prompting methods and do not distinguish between input and output tokens. While these simplifications may be justified when few-shot examples dominate the token count, a more precise breakdown of the components is necessary for accurate token efficiency analysis in concise prompts.

We present further discussion of token efficiency in Appendix A.

Token Analysis To show the token efficiency of the batch prompts we are using, we present an analysis of the token counts of batch prompts and standard prompts in Figure 5 for the RTE task.³

³Prior work creating batch prompts for RTE did not provide complete batch prompts containing in-context examples

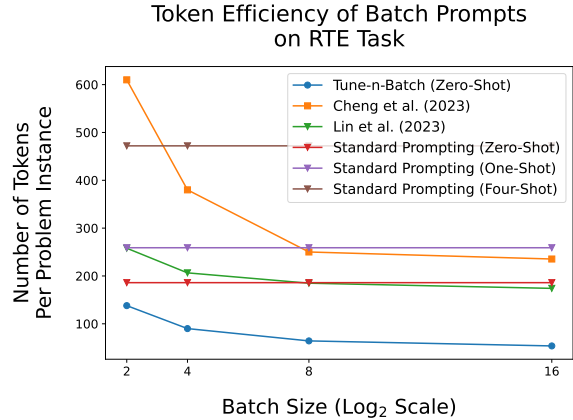


Figure 5: Comparing the token efficiency of batch prompting and standard prompting on the RTE dataset.

As seen in Figure 5, Tune-n-Batch batch prompts are substantially more token efficient than previous batch prompting and standard prompting methods. Across all batch sizes greater than 1, the number of tokens per problem instance was significantly lower for our method than that of other prompting methods. Moreover, the number of tokens per query in our batch prompts decreased inversely with the batch size.

When comparing this method to prior batch prompting methods, we see how the removal of in-context examples have a dramatic effect on reducing the overall prompt length. Through the Tune-

across batch sizes or code for forming the batch prompts. The token count estimates for existing batch prompts on are based on assumptions of example formatting and the number of in-context examples.

n-Batch method, we do not require these examples to induce correct responses to batch prompts, as we use fine-tuning to teach LLMs to do this. As the batch size increases, we can expect the number of tokens per problem instance to eventually converge among all three methods, but the batch size required for this to occur is not practically applicable. Moreover, these methods reported decreases in accuracy as batch size increases unlike the results we present in which accuracy remains relatively stable across batch sizes for most tasks.

5 Related Work

Batching in Deep Learning Systems Batching has been employed in neural network training since Rumelhart et al. (1986) and has since gained popularity for enhancing the efficiency of both training and inference (Bertsekas, 2011; Bengio, 2012). It has become a standard practice for training large language models since the introduction of the transformer architecture (Vaswani et al., 2017; Brown et al., 2020; Yu et al., 2022; Touvron et al., 2023). Under the right conditions, batching can accelerate training through parallelization (Goyal et al., 2018), improve accuracy by aggregating gradients (Masters and Luschi, 2018), and reduce environmental costs (Yarally et al., 2023).

Batching has also been essential for reducing inference time when processing large sets of model queries. However, the parameter count of large language models (LLMs) is growing faster than GPU memory capacities (Rajbhandari et al., 2021), challenging the traditional batching approach where each query is allocated separate GPU memory and processed in parallel.

SysML approaches to redundant input While Cheng et al. (2023), Lin et al. (2023), and this work approach the concept of reusing redundant inputs to an LLM through efficient prompting strategies, there are also approaches that attempt to solve this problem at a lower level. KV cache reusing schemes reduces the time it takes to process input text in an LLM by pre-computing and caching the tensors of frequently reused texts. Yao et al. (2024) improves upon prior methods, which often trade-off speed with generation quality, by selectively re-computing certain portions of the text while reusing the rest from the KV cache, thereby preserving cross-attention and thus the generation quality while still benefiting from the speedup provided by caching. Another approach, called Attention-

Store, uses a hierarchical KV caching system to reuse input tensors across multi-turn conversations in LLMs (Gao et al., 2024).

Fine-tuning for Instruction Following Our methodology of fine-tuning for batch prompting on certain tasks while expecting the model to generalize this style of prompting on other unseen tasks is motivated by prior work on instruction following. The zero-shot capabilities of LLMs (Brown et al., 2020) drove Sanh et al. (2022) to discover that explicitly fine-tuning an LLM on a large set of tasks could lead to better zero-shot generalization on unseen tasks, implying that LLMs can be trained on how to follow instructions. Wei et al. (2022) further corroborates this phenomenon by developing an instruction-tuned model called FLAN, which outperformed GPT-3 on several tasks despite its smaller size.

6 Conclusion

In this work, we present a novel approach to batch prompting, demonstrating that fine-tuning a large language model on a diverse dataset of batch prompts enables effective batch prompting without the need for lengthy prompts or few-shot examples. Our fine-tuned model maintains stable performance across various batch sizes, even on unseen tasks, contrasting with the performance degradation observed in non-fine-tuned models as batch size increases. Our findings demonstrate that fine-tuning LLMs for batch prompting can significantly improve their efficiency and applicability in real-world scenarios, making it possible to process a larger number of queries using fewer computational resources.

We encourage future work to explore batch prompting as an efficient alternative to standard prompting methods. Future work on batch prompting should incorporate chain-of-thought (CoT) reasoning (Wei et al., 2023), as CoT could be instrumental for extending batch prompting to multi-step tasks, such as mathematical reasoning. While this paper mainly explored single task batch prompts, the development of well-performing batch prompts encompassing multiple tasks could offer flexibility while retaining the efficiency gains from batch prompting. With these efficiency benefits in mind, we believe future work should also focus on developing large language models capable of processing batch prompts with large batch sizes as accurately as standard prompts.

543 Limitations

544 Our study has several limitations that should be ad-
545 dressed in future research. We only experimented
546 with a single model, LLaMA-3-8B-Instruct, due
547 the computational requirements of fine-tuning
548 larger models. Future work should investigate
549 whether our findings generalize to a diverse set of
550 models of different sizes, architectures, and capabil-
551 ities, such as T5 and Llama-3-70B-Instruct, to de-
552 termine if effective batch prompting is an emergent
553 property of larger models or if it can be achieved
554 after fine-tuning with smaller models as well.

555 Because of limited context window sizes and
556 the token scaling of including larger batches in a
557 context, our analysis only went up to batches of
558 size 32. Further work is needed to determine if the
559 performance stability observed in our experiments
560 extends to larger batch sizes of ≥ 64 .

561 Another limitation of our study is the potential
562 loss of generalizability due to fine-tuning. When
563 comparing standard prompting to batch prompting,
564 we observed a decrease in performance on tasks
565 that were not included in the fine-tuning process.
566 Although we attempted to mitigate this risk by fine-
567 tuning on a small number of examples, future work
568 should explore alternative methods for maintain-
569 ing generalizability, such as meta-learning, multi-
570 task learning, and fine-tuning approaches designed
571 to prevent the loss of generality. Exploring the
572 use of instruction tuning (Zhang et al., 2024) and
573 Reinforcement Learning from Human Feedback
574 (RLHF) (Ouyang et al., 2022) or Direct Preference
575 Optimization (DPO) (Rafailov et al., 2023) to fine-
576 tune the model for batch prompting could be a
577 viable alternative to Parameter-efficient fine-tuning
578 (PEFT). Additionally, more extensive experiments
579 are needed to quantify the extent of catastrophic for-
580 getting and its impact on the model’s performance
581 on non-batch prompting tasks after fine-tuning.

582 References

583 Rishabh Agarwal, Avi Singh, Lei M. Zhang, Bernd
584 Bohnet, Luis Rosias, Stephanie Chan, Biao Zhang,
585 Ankesh Anand, Zaheer Abbas, Azade Nova, John D.
586 Co-Reyes, Eric Chu, Feryal Behbahani, Aleksandra
587 Faust, and Hugo Larochelle. 2024. [Many-shot in-
588 context learning](#). *Preprint*, arXiv:2404.11018.

589 AI@Meta. 2024. [Llama 3 model card](#).

590 Yuntao Bai, Saurav Kadavath, Sandipan Kundu,
591 Amanda Askell, Jackson Kernion, Andy Jones, Anna
592 Chen, Anna Goldie, Azalia Mirhoseini, Cameron

McKinnon, Carol Chen, Catherine Olsson, Christo- 593
pher Olah, Danny Hernandez, Dawn Drain, Deep 594
Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, 595
Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua 596
Landau, Kamal Ndousse, Kamile Lukosuite, Liane 597
Lovitt, Michael Sellitto, Nelson Elhage, Nicholas 598
Schiefer, Noemi Mercado, Nova DasSarma, Robert 599
Lasenby, Robin Larson, Sam Ringer, Scott John- 600
ston, Shauna Kravec, Sheer El Showk, Stanislav Fort, 601
Tamera Lanham, Timothy Telleen-Lawton, Tom Con- 602
erly, Tom Henighan, Tristan Hume, Samuel R. Bow- 603
man, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, 604
Nicholas Joseph, Sam McCandlish, Tom Brown, and 605
Jared Kaplan. 2022. [Constitutional ai: Harmlessness
606 from ai feedback](#). *Preprint*, arXiv:2212.08073. 607

Yoshua Bengio. 2012. [Practical recommendations
608 for gradient-based training of deep architectures](#).
609 *Preprint*, arXiv:1206.5533. 610

Amanda Bertsch, Maor Ivgi, Uri Alon, Jonathan Berant,
Matthew R. Gormley, and Graham Neubig. 2024. [In-context learning with long-context models: An
611 in-depth exploration](#). *Preprint*, arXiv:2405.00200. 612
613
614

Dimitri P Bertsekas. 2011. Incremental gradient, sub- 615
gradient, and proximal methods for convex optimiza- 616
tion: A survey. 617

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie 618
Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind 619
Neelakantan, Pranav Shyam, Girish Sastry, Amanda 620
Askell, Sandhini Agarwal, Ariel Herbert-Voss, 621
Gretchen Krueger, Tom Henighan, Rewon Child, 622
Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, 623
Clemens Winter, Christopher Hesse, Mark Chen, 624
Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin 625
Chess, Jack Clark, Christopher Berner, Sam Mc- 626
Candlish, Alec Radford, Ilya Sutskever, and Dario 627
Amodei. 2020. [Language models are few-shot learn-
628 ers](#). *Preprint*, arXiv:2005.14165. 629

Zhoujun Cheng, Jungo Kasai, and Tao Yu. 2023. [Batch
630 prompting: Efficient inference with large language
631 model apis](#). *Preprint*, arXiv:2301.08721. 632

William B. Dolan and Chris Brockett. 2005. [Automati-
633 cally constructing a corpus of sentential paraphrases](#).
634 In *Proceedings of the Third International Workshop
635 on Paraphrasing (IWP2005)*. 636

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang,
Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou
Yu, and Pengfei Zuo. 2024. [Attentionstore: Cost-
637 effective attention reuse across multi-turn conversa-
638 tions in large language model serving](#). *arXiv preprint
639 arXiv:2403.19708*. 640
641
642

Sealey Gosset. 1908. [The probable error of a mean](#). 643
Biometrika, 6(1):1–25. 644

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter No- 645
ordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew 646
Tulloch, Yangqing Jia, and Kaiming He. 2018. [Ac-
647 curate, large minibatch sgd: Training imagenet in 1
648 hour](#). *Preprint*, arXiv:1706.02677. 649

650	Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan	Taewoon Kim, Gunjan Chhablani, Nihal Nayak, De-	704
651	Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and	bajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang,	705
652	Weizhu Chen. 2021. Lora: Low-rank adaptation of	Han Wang, Matteo Manica, Sheng Shen, Zheng Xin	706
653	large language models . <i>Preprint</i> , arXiv:2106.09685.	Yong, Harshit Pandey, Rachel Bawden, Thomas	707
654	Shankar Iyer, Nikhil Dandekar, and Kornél Csernai.	Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma,	708
655	2017. First quora dataset release: Question pairs .	Andrea Santilli, Thibault Fevry, Jason Alan Fries,	709
656	Online. Accessed: YYYY-MM-DD.	Ryan Teehan, Tali Bers, Stella Biderman, Leo Gao,	710
657	Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang,	Thomas Wolf, and Alexander M. Rush. 2022. Multi-	711
658	and Eduard Hovy. 2017. Race: Large-scale reading	task prompted training enables zero-shot task gener-	712
659	comprehension dataset from examinations . <i>Preprint</i> ,	alization . <i>Preprint</i> , arXiv:2110.08207.	713
660	arXiv:1704.04683.	Guijin Son, Sangwon Baek, Sangdae Nam, Ilgyun	714
661	Jianzhe Lin, Maurice Diesendruck, Liang Du, and	Jeong, and Seungone Kim. 2024. Multi-task infer-	715
662	Robin Abraham. 2023. Batchprompt: Accomplish	ence: Can large language models follow multiple	716
663	more with less . <i>Preprint</i> , arXiv:2309.00384.	instructions at once? <i>Preprint</i> , arXiv:2402.11597.	717
664	Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape,	Charles Spearman. 1961. The proof and measurement	718
665	Michele Bevilacqua, Fabio Petroni, and Percy	of association between two things.	719
666	Liang. 2023. Lost in the middle: How language mod-	Alon Talmor, Jonathan Herzig, Nicholas Lourie, and	720
667	els use long contexts . <i>Preprint</i> , arXiv:2307.03172.	Jonathan Berant. 2019. Commonsenseqa: A question	721
668	Dominic Masters and Carlo Luschi. 2018. Revisiting	answering challenge targeting commonsense knowl-	722
669	small batch training for deep neural networks .	edge . <i>Preprint</i> , arXiv:1811.00937.	723
670	<i>Preprint</i> , arXiv:1804.07612.	Hugo Touvron, Louis Martin, Kevin Stone, Peter Al-	724
671	Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Car-	bert, Amjad Almahairi, Yasmine Babaei, Nikolay	725
672	roll L. Wainwright, Pamela Mishkin, Chong Zhang,	Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti	726
673	Sandhini Agarwal, Katarina Slama, Alex Ray, John	Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton	727
674	Schulman, Jacob Hilton, Fraser Kelton, Luke Miller,	Ferrer, Moya Chen, Guillem Cucurull, David Esiobu,	728
675	Maddie Simens, Amanda Askell, Peter Welinder,	Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller,	729
676	Paul Christiano, Jan Leike, and Ryan Lowe. 2022.	Cynthia Gao, Vedanuj Goswami, Naman Goyal, An-	730
677	Training language models to follow instructions with	thony Hartshorn, Saghar Hosseini, Rui Hou, Hakan	731
678	human feedback . <i>Preprint</i> , arXiv:2203.02155.	Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa,	732
679	F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel,	Isabel Kloumann, Artem Korenev, Punit Singh Koura,	733
680	B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer,	Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Di-	734
681	R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,	ana Liskovich, Yinghai Lu, Yuning Mao, Xavier Mar-	735
682	D. Cournapeau, M. Brucher, M. Perrot, and E. Duch-	tinet, Todor Mihaylov, Pushkar Mishra, Igor Moly-	736
683	esnay. 2011. Scikit-learn: Machine learning in	bog, Yixin Nie, Andrew Poulton, Jeremy Reizen-	737
684	Python. <i>Journal of Machine Learning Research</i> ,	stein, Rashi Rungta, Kalyan Saladi, Alan Schelten,	738
685	12:2825–2830.	Ruan Silva, Eric Michael Smith, Ranjan Subrama-	739
686	Rafael Rafailov, Archit Sharma, Eric Mitchell, Christo-	nian, Xiaoqing Ellen Tan, Binh Tang, Ross Tay-	740
687	pher D Manning, Stefano Ermon, and Chelsea Finn.	lor, Adina Williams, Jian Xiang Kuan, Puxin Xu,	741
688	2023. Direct preference optimization: Your language	Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan,	742
689	model is secretly a reward model . In <i>Advances in</i>	Melanie Kambadur, Sharan Narang, Aurelien Rod-	743
690	<i>Neural Information Processing Systems</i> , volume 36,	riguez, Robert Stojnic, Sergey Edunov, and Thomas	744
691	pages 53728–53741. Curran Associates, Inc.	Scialom. 2023. Llama 2: Open foundation and fine-	745
692	Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley,	tuned chat models . <i>Preprint</i> , arXiv:2307.09288.	746
693	Shaden Smith, and Yuxiong He. 2021. Zero-infinity:	Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob	747
694	Breaking the gpu memory wall for extreme scale	Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz	748
695	deep learning . <i>Preprint</i> , arXiv:2104.07857.	Kaiser, and Illia Polosukhin. 2017. Attention is all	749
696	David E Rumelhart, Geoffrey E Hinton, and Ronald J	you need. <i>Advances in neural information processing</i>	750
697	Williams. 1986. Learning representations by back-	<i>systems</i> , 30.	751
698	propagating errors. <i>nature</i> , 323(6088):533–536.	Alex Wang, Amanpreet Singh, Julian Michael, Felix	752
699	Victor Sanh, Albert Webson, Colin Raffel, Stephen H.	Hill, Omer Levy, and Samuel R. Bowman. 2019.	753
700	Bach, Lintang Sutawika, Zaid Alyafeai, Antoine	Glue: A multi-task benchmark and analysis plat-	754
701	Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja,	form for natural language understanding . <i>Preprint</i> ,	755
702	Manan Dey, M Saiful Bari, Canwen Xu, Urmish	arXiv:1804.07461.	756
703	Thakker, Shanya Sharma Sharma, Eliza Szczechla,	Alex Warstadt, Amanpreet Singh, and Samuel R Bow-	757
		man. 2019. Cola: The corpus of linguistic acceptabil-	758
		ity (with added annotations).	759

760 Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin
761 Guu, Adams Wei Yu, Brian Lester, Nan Du, An-
762 drew M. Dai, and Quoc V. Le. 2022. [Finetuned](#)
763 [language models are zero-shot learners](#). *Preprint*,
764 arXiv:2109.01652.

765 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten
766 Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and
767 Denny Zhou. 2023. [Chain-of-thought prompting elic-](#)
768 [its reasoning in large language models](#). *Preprint*,
769 arXiv:2201.11903.

770 Adina Williams, Nikita Nangia, and Samuel R. Bow-
771 man. 2018. [A broad-coverage challenge corpus for](#)
772 [sentence understanding through inference](#). *Preprint*,
773 arXiv:1704.05426.

774 Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang,
775 Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi
776 Rungta, Karthik Abinav Sankararaman, Barlas Oguz,
777 et al. 2023. Effective long-context scaling of founda-
778 tion models. *arXiv preprint arXiv:2309.16039*.

779 Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua
780 Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and
781 Junchen Jiang. 2024. Cacheblend: Fast large lan-
782 guage model serving with cached knowledge fusion.
783 *arXiv preprint arXiv:2405.16444*.

784 Tim Yarally, Luís Cruz, Daniel Feitosa, June Sallou,
785 and Arie van Deursen. 2023. [Batching for green](#)
786 [ai – an exploratory study on inference](#). *Preprint*,
787 arXiv:2307.11434.

788 Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soo-
789 jeong Kim, and Byung-Gon Chun. 2022. Orca: A
790 distributed serving system for {Transformer-Based}
791 generative models. In *16th USENIX Symposium*
792 *on Operating Systems Design and Implementation*
793 *(OSDI 22)*, pages 521–538.

794 Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang,
795 Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tian-
796 wei Zhang, Fei Wu, and Guoyin Wang. 2024. [In-](#)
797 [struction tuning for large language models: A survey](#).
798 *Preprint*, arXiv:2308.10792.

A Batch Prompting Token Efficiency

Table 2 breaks down the components of standard prompts (SP) and batch prompts (BP), providing formulas to calculate input and output token efficiency. The table reveals that SP and BP have relatively equal output token efficiency, with SP slightly more efficient due to not requiring answer index formatting. However, this difference may be significant given the higher API costs for output generation compared to input tokens. The task description T , formatting instructions F , shared context C , and examples K are fixed costs across all batch sizes, while query tokens Q depend on the batch size. In scenarios where fixed costs are a significant portion of the total tokens, batch prompting can be substantially more efficient than standard prompting. As batch size increases, fixed costs are amortized over more queries, reducing per-query token costs. This is particularly beneficial when fixed costs are high relative to variable costs. However, the benefits of batch prompting may diminish beyond a certain batch size, as variable costs scale linearly while fixed costs remain constant, leading to a point of diminishing returns.

B Supplemental Results

Table 3 provides the numerical values for accuracy present in Figure 3 along with the MNLI Matched task.

C Licensing and Artifact Information

Our use of LLaMA-3-8B-Instruct is permitted under research purposes. The use of existing datasets, such as those in the batch prompting dataset, is allowed for research and non-commercial purposes in accordance with the applicable data usage agreements.

D Hyperparameter Settings

We used the default hyperparameters for SFT training for LORA. We used LoRA (Low-Rank Adaptation) with default values of $lora_alpha = 16$, $lora_dropout = 0.1$, and $r=64$. Hyperparameters like a learning rate of 2×10^{-4} weight decay of 0.001, and max gradient norm of 0.3 were employed during training.

For parsing and regular expression testing, we used Python’s built-in `re` library with functions like `re.findall()`, `re.finditer()`, and `re.search()`, along with various regular expression patterns

tailored to the task requirements. For testing and evaluation, we employed the scikit-learn library (Pedregosa et al., 2011), utilizing modules like `sklearn.metrics` for computing evaluation metrics, `sklearn.model_selection` for techniques like cross-validation, and `sklearn.preprocessing` for data normalization and scaling. The specific implementations and parameter settings were adjusted based on the parsing and evaluation needs.

Standard Prompting		Batch Prompting	
T - Task Description (T_{SP})		T - Task Description (T_{BP})	
F - Formatting (F_{SP})		F - Formatting (F_{BP})	
C - Context (opt) (C_{SP})		C - Context (opt) (C_{BP})	
K - Examples (opt) (K_{SP})		K - Examples (opt) (K_{BP})	
Q - Query (Q_{SP})		b · Q - Query tokens per query (Q_{BP})	
A - Answer (A_{SP})		b · A - Answer tokens per answer (A_{BP})	
Total Token Formula (T_{total})			
$T_{total,SP} = T_{SP} + F_{SP} + C_{SP} + K_{SP} + Q_{SP} + A_{SP}$		$T_{total,BP} = T_{BP} + F_{BP} + C_{BP} + K_{BP} + b(Q_{BP} + A_{BP})$	
Input Token Formula (T_{input})			
$T_{input,SP} = T_{SP} + F_{SP} + C_{SP} + K_{SP} + Q_{SP}$		$T_{input,BP} = T_{BP} + F_{BP} + C_{BP} + K_{BP} + b(Q_{BP})$	
Output Token Formula (T_{output})			
$T_{output,SP} = A_{SP}$		$T_{output,BP} = b(A_{BP})$	

Table 2: Breaking down the components of prompts for token efficiency analysis.

Task	Non-Fine-Tuned							Tune-n-Batch								
	Prompting Type and Batch Size								Prompting Type and Batch Size							
	Seen	SP=1	BP=2	BP=4	BP=8	BP=16	BP=32	Seen	SP=1	BP=2	BP=4	BP=8	BP=16	BP=32		
CoLA	No	80.4	79.0	77.2	77.5	66.7	76.9	Yes	77.6	81.4	80.3	80.5	79.8	80.3		
QQP	No	68.8	64.6	63.6	62.5	61.5	54.1	Yes	71.6	68.4	71.7	71.7	71.3	71.4		
SST2	No	90.4	91.0	92.4	92.0	91.4	93.2	Yes	93.6	94.6	94.5	95.2	94.7	96.4		
Common Sense	No	59.2	61.6	61.1	60.6	50.2	31.8	No	64.0	50.6	54.7	53.5	52.6	53.7		
MNLI Matched	No	62.8	56.0	53.7	51.7	49.0	44.6	No	57.6	47.4	48.5	45.3	48.5	44.3		
MNLI Mismatched	No	64.0	59.8	54.0	54.2	51.5	47.2	No	61.2	45.0	49.4	47.7	48.9	49.1		
MRPC	No	75.6	73.8	72.8	69.6	65.5	58.6	No	68.8	64.5	66.7	64.7	65.3	64.9		
RACE	No	64.2	61.0	51.7	54.9	-	-	No	64.8	62.4	66.7	62.0	-	-		
RTE	No	68.0	65.6	67.8	64.0	58.1	68.8	No	79.2	77.5	79.3	80.6	80.2	78.7		

Table 3: Comparing the Performance of Llama3-8B Instruct before and after fine-tuning. All other tasks report the performance with the task excluded from training. SP denotes Standard Prompting, whereas BP represents Batch Prompting. Seen indicates that the training set of the task was included in fine-tuning.

855 **E Prompt Comparisons**

856 We include our prompts as well as few-shot batch prompts from prior work. We see that the few-shot
857 batch prompts require a significant number of examples and additional text to demonstrate the formatting
858 of the task, which is not required for our fine-tuned model.

Our Zero-Shot Standard Prompt for QQP

Your task is to determine if a pair of questions from the QQP dataset are duplicates. Classify each question pair as 0 (not duplicate) or 1 (duplicate) by analyzing lexical similarity and question intent. Answer using the format "The classification is <calculated_answer>", where <calculated_answer> is 0 or 1. Do not output any other text, such as intermediate reasoning, other than The classification is <calculated_answer> for the question pair asked.

Question1: What is the meaning of the Urdu word 'Jaah'?

Question2: What is the meaning of Urdu word 'Ziadi'?

859

Our Zero-Shot Batch Prompt for QQP

Your task is to determine if pairs of questions are duplicates from a list of question pairs from the QQP dataset. Classify each question pair as 0 (not duplicate) or 1 (duplicate). Analyze lexical similarity and question intent. For each question pair, answer using the exact format "Answer[index]: The classification is <calculated_answer>", where 'index' is the corresponding question index being answered, and <calculated_answer> is a 0 or 1. Do not output any other text, such as intermediate reasoning, other than Answer[index]: The answer is <calculated_answer> for each question-pair asked.

Question1[0]: What is the meaning of the Urdu word 'Jaah'?

Question2[0]: What is the meaning of Urdu word 'Ziadi'?

Question1[1]: How can we simplify our life?

Question2[1]: Life Advice: How can I make my life simpler?

Question1[2]: How does Venmo work?

Question2[2]: Are Venmo payouts reversible?

Question1[3]: What should I know before buying a house?

Question2[3]: What should I know before buying a house in Europe?

860

(Lin et al., 2023) Few-Shot Batch Prompt for QQP

You are a professional NLP expert at duplicate question detection. You will be given [BATCH-SIZE] pairs of data from Quora Question Pairs (QQP) dataset each time, as input. Each data includes a pair data, "Question1" and "Question2". Your goal to determine whether two questions are duplicates of each other. You need to classify into below two classes:
class 1: if they have the same meaning (semantically equivalent).
class 0: if they do NOT have the same meaning.

=====
Question pair 0:
Question1: xxxxx
Question2: xxxxx
Question pair 1:
Question1: xxxxx

861

Question2: xxxxx

.....

=====

Below are the outputs you need to generate. "X" can be '1' or '0'. [Conf-Description]

=====

Label for Question pair 0: [class X][Place-Holder-Conf]

Label for Question pair 1: [class X][Place-Holder-Conf]

.....

=====

Follow the formatting of the following examples.

Question pair 0:

Question1: Does Hanes's online store accept PayPal?

Question2: How are Hanes t-shirts made?

Question pair 1:

Question1: What are the best Norditrac exercise routines?

Question2: What are the best exercise routines?

Question pair 2:

Question1: How does someone sever their carotid artery with a kitchen knife?

Question2: Do any muscle protect the carotid artery or is it right under the skin?

Question pair 3:

Question1: Gravity: Why doesn't the Earth fall into the Sun or the Moon fall into the Earth?

Question2: Why doesn't the earth accelerate towards the sun?

Label for Question pair 0: [0](Confident)

Label for Question pair 1: [0](Not Confident)

Label for Question pair 2: [0](Confident)

Label for Question pair 3: [1](Confident)

Now answer the following questions.

=====

Question pair 0:

Question1: What is the meaning of the Urdu word 'Jaah'?

Question2: What is the meaning of Urdu word 'Ziadi'?

Question pair 1:

Question1: How can we simplify our life?

Question2: Life Advice: How can I make my life simpler?

Question pair 2:

Question1: How does Venmo work?

Question2: Are Venmo payouts reversible?

Question pair 3:

Question1: What should I know before buying a house?

Question2: What should I know before buying a house in Europe?

=====

Below are the outputs you need to generate. "X" can be '1' or '0'. [Conf-Description]

=====

Label for Question pair 0: [class X][Place-Holder-Conf]

Label for Question pair 1: [class X][Place-Holder-Conf]

Label for Question pair 2: [class X][Place-Holder-Conf]

Label for Question pair 3: [class X][Place-Holder-Conf]

.....

=====

Please make sure each generated label is in format of [class X].
Please make sure to generate [BATCH-SIZE] labels.

(Note: (Lin et al., 2023) only provided a condensed version of the batch prompt for the QQP dataset, omitting few-shot examples and other text instructions. We inferred a likely prompt based on the abbreviated version included in the appendix, but there may be minor differences from the one used by the original authors.)

Our Zero-Shot Standard Prompt for SST2

Your task is to classify a sentence from the SST-2 dataset as positive or negative in sentiment. Answer using the format "The answer is <calculated_answer>.", where <calculated_answer> is 0 for negative sentiment and 1 for positive sentiment. Do not output any other text, such as intermediate reasoning, other than The answer is <calculated_answer> for the question asked.
Sentence: that stealing harvard is too busy getting in its own way to be anything but frustrating , boring , and forgettable

Our Zero-Shot Batch Prompt for SST2

Your task is to classify sentences for each question in a list of questions from the SST-2 dataset as positive or negative in sentiment. For each question, notated as Question[index], answer using the the exact format "Answer[index]: The answer is <calculated_answer>.", where 'index' is the corresponding question number being answered, and <calculated_answer> is 0 for negative sentiment and 1 for positive sentiment. Do not output any other text, such as intermediate reasoning, other than Answer[index]: The answer is <calculated_answer> for each question asked.

Question[0]: Sentence[0]: that stealing harvard is too busy getting in its own way to be anything but frustrating , boring , and forgettable
Question[1]: Sentence[1]: everyone 's insecure in lovely and amazing , a poignant and wryly amusing film about mothers , daughters and their relationships.
Question[2]: Sentence[2]: good news
Question[3]: Sentence[3]: been saved if the director , tom dey , had spliced together bits and pieces of midnight run and 48 hours (and , for that matter , shrek)

(Cheng et al., 2023) Few-Shot Batch Prompt for GSM8K

System

The answers should respond to each question the user inputs. One line one answer starting with A[idx],no other line break.

User

Q[1]: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

Q[2]: If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?

Q[3]: Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?

Q[4]: Jason had 20 lollipops. He gave Denny some lollipops. Now Jason has 12 lollipops. How many lollipops did Jason give to Denny?

Assistant

A[1]: There are 15 trees originally. Then there were 21 trees after some more were planted. So

there must have been $21 - 15 = 6$. The answer is 6.

A[2]: There are originally 3 cars. 2 more cars arrive. $3 + 2 = 5$. The answer is 5.

A[3]: Originally, Leah had 32 chocolates. Her sister had 42. So in total they had $32 + 42 = 74$. After eating 35, they had $74 - 35 = 39$. The answer is 39.

A[4]: Jason started with 20 lollipops. Then he had 12 after giving some to Denny. So he gave Denny $20 - 12 = 8$. The answer is 8.

User

Q[1]: Shawn has five toys. For Christmas, he got two toys each from his mom and dad. How many toys does he have now?

Q[2]: There were nine computers in the server room. Five more computers were installed each day, from monday

to thursday. How many computers are now in the server room? Q[3]: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On wednesday, he lost 2 more. How many golf balls did he have at the end of wednesday?

Q[4]: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?

Assistant

A[1]: Shawn started with 5 toys. If he got 2 toys each from his mom and dad, then that is 4 more toys. $5 + 4 = 9$. The answer is 9.

A[2]: There were originally 9 computers. For each of 4 days, 5 more computers were added. So $5 * 4 = 20$ computers were added. $9 + 20$ is 29. The answer is 29.

A[3]: Michael started with 58 golf balls. After losing 23 on tuesday, he had $58 - 23 = 35$. After losing 2 more, he had $35 - 2 = 33$ golf balls. The answer is 33.

A[4]: Olivia had 23 dollars. 5 bagels for 3 dollars each will be $5 * 3 = 15$ dollars. So she has $23 - 15$ dollars left. $23 - 15$ is 8. The answer is 8.

User

Q[1]: A garden produced 237 potatoes, 60 fewer cucumbers and twice as many peppers than the cucumbers. How many vegetables did the garden produce?

Q[2]: John's cow weighs 400 pounds. It increased its weight to 1.5 times its starting weight. He is able to sell the cow for \$3 per pound. How much more is it worth after gaining the weight?

Q[3]: John writes 20 pages a day. How long will it take him to write 3 books that are 400 pages each?

Q[4]: James has a rainwater collection barrel. For each inch of rain he collects 15 gallons. On Monday it rained 4 inches and on Tuesday it rained 3 inches. He can sell water for \$1.2 per gallon. How much money did he make from selling all the water?

Assistant

A[1]: The garden produced $237 - 60 = 177$ cucumbers. The garden produced $177 * 2 = 354$ peppers. The garden produced $237 + 177 + 354 = 768$ vegetables. The answer is 768.

A[2]: The cow initially weighs $400 * 1.5 = 600$ pounds. So it gained $600 - 400 = 200$ pounds. It is worth $200 * 3 = 600$ dollars more. The answer is 600.

A[3]: He wants to write $3 * 400 = 1200$ pages. So it will take him $1200 / 20 = 60$ days The answer is 60.

A[4]: It rained $3 + 4 = 7$ inches So he collected $7 * 15 = 105$ gallons So he makes $105 * 1.2 = 126$ from selling the water. The answer is 126.

User

Q[0]: To get to an island called "Virgo", Tom needs to travel by plane and then by boat. The plane trip is four times longer than the boat trip, and the boat trip takes up to 2 hours. In how many hours is Tom able to get to the "Virgo" island?

Q[1]: Winwin won \$50 in a lottery. She paid 20% for the tax and she paid \$5 for the processing fee. How much was she able to take home?

Q[2]: Grandma left \$124,600 in her will. She gave half of it to her favorite grandchild, Shelby.

The rest was to be evenly divided among the remaining 10 grandchildren. How many
Q[3]: John and his best friend Steve bought 12 cupcakes together. Each cupcake cost \$1.50. If they split the costs evenly, how much did each person pay?
Assistant