

How Optimal is Greedy Decoding for Extractive Question Answering?

Or Castel¹

Ori Ram¹

Avia Efrat¹

Omer Levy^{1,2}

OR.CASTEL@CS.TAU.AC.IL

ORI.RAM@CS.TAU.AC.IL

AVIA.EFRAT@CS.TAU.AC.IL

LEVYOMER@CS.TAU.AC.IL

¹Blavatnik School of Computer Science, Tel Aviv University

²Meta AI Research

Abstract

Fine-tuned language models use greedy decoding to answer reading comprehension questions with relative success. However, this approach does not ensure that the answer is a span in the given passage, nor does it guarantee that it is the most probable one. Does greedy decoding actually perform worse than an algorithm that *does* adhere to these properties? To study the performance and optimality of greedy decoding, we present *exact-extract*, a decoding algorithm that efficiently finds the most probable answer span in the passage. We compare the performance of T5 with both decoding algorithms on zero-shot and few-shot extractive question answering. When no training examples are available, *exact-extract* significantly outperforms greedy decoding. However, greedy decoding quickly converges towards the performance of *exact-extract* with the introduction of a few training examples, becoming more extractive and increasingly likelier to generate the most probable span as the training set grows. We also show that self-supervised training can bias the model towards extractive behavior, increasing performance in the zero-shot setting without resorting to annotated examples. Overall, our results suggest that pretrained language models are so good at adapting to extractive question answering, that it is often enough to fine-tune on a small training set for the greedy algorithm to emulate the optimal decoding strategy.¹

1. Introduction

Extractive question answering is the task of answering a question given a passage, assuming the answer appears as a span in the passage. It is a main component in state-of-the-art methods for open-domain question answering [Karpukhin et al., 2020, Izacard and Grave, 2021], and can facilitate numerous other NLP tasks, such as relation extraction [Levy et al., 2017], coreference resolution [Wu et al., 2020], named entity recognition [Li et al., 2020], and more. Generative language models usually address this task via greedy decoding algorithms, which do not guarantee two key properties: (1) they are not *extractive*, i.e. they can produce texts that are not spans in the passage, (2) they are not *exact*, i.e. they do not necessarily generate the most probable output according to the model. In this work, we show that despite lacking any formal guarantees, greedy decoding can approach the performance of a theoretically optimal decoding algorithm across a variety of extractive question answering benchmarks, even when only a few training examples are available.

To that end, we introduce *exact-extract*, a decoding algorithm that efficiently calculates the model-assigned probabilities of all spans in the passage, allowing us to (provably) select the most probable span. We compare greedy decoding with *exact-extract* on the recently-proposed few-shot

1. Our code and models are publicly available at: <https://github.com/ocastel/exact-extract>



Figure 1: Performance of T5-large on SQuAD when using greedy (green) and optimal (red) decoding, given different amounts of training examples. As the amount of examples increases, the performance gap between the decoding algorithms diminishes.

question answering benchmark [Ram et al., 2021], which contains subsampled training sets of 16 to 1024 examples from 8 different datasets. Specifically, we fine-tune a pretrained language model, T5-large [Raffel et al., 2020], and measure the performance of both decoding algorithms.

In the zero-shot setting, where no annotated examples are available, there is a significant performance margin (11.3 F1 points on average across datasets) between greedy decoding and exact-extract. This gap quickly shrinks as more annotated examples are introduced; even 16 training examples are enough to narrow the average performance margin to 2.8 points, with 1024 examples diminishing it to 0.3. Figure 1 shows this trend on the SQuAD dataset [Rajpurkar et al., 2016].

We further measure how often greedy decoding generates spans from the given passage (i.e. the algorithm’s *extractiveness*), and observe a strong correlation between extractiveness and performance. In particular, we notice that in the zero-shot setting, where exact-extract strongly outperforms greedy decoding, the greedy algorithm is substantially less extractive. To increase extractiveness, we propose an additional self-supervised pretraining phase inspired by recurring span selection [Ram et al., 2021]. Training with this objective enhances the model’s tendency to generate answers from the context, and consequentially improves the performance of greedy decoding in this challenging setting.

Overall, our results demonstrate that although greedy decoding does not explicitly guarantee either extractiveness or exactness, the underlying model (T5) adapts so well to the task of extractive question answering, that even a few examples are enough to allow the naive greedy decoding algorithm to generate answers that rival those of an optimal decoding algorithm.

2. Problem Setting

The task of *extractive question answering* (extractive QA) [Rajpurkar et al., 2016] is to select a span a from a given passage T that answers a question Q . In this work, we focus on few-shot and zero-shot extractive QA [Ram et al., 2021], where the learner is given a small set of training and development examples (from 16 to 1024 examples), or none at all. These settings resemble real-world use-cases, where an abundance of data is not necessarily available, as annotating large datasets is expensive and may even require domain and language experts.

Extractive QA is typically modeled via span selection models that point to the start and end of the answer span [Seo et al., 2018, Devlin et al., 2019]. This approach is *extractive*,² and also allows for *exact*³ decoding since it computes a score for every possible span.

However, a recent trend in NLP is to frame all tasks as text-to-text [Raffel et al., 2020, Brown et al., 2020]. Indeed, various conditional language models have shown strong results on extractive QA [Raffel et al., 2020, Lewis et al., 2020, Chada and Natarajan, 2021], generating answers via greedy decoding and its variants. But how often does greedy decoding violate these properties *in practice*, and does it actually affect its performance?

3. The Exact-Extract Algorithm

To study the greedy decoding algorithm in the context of extractive QA, we compare it to a new algorithm that produces the optimal extractive decoding (i.e. the span with the highest probability) from an autoregressive model: *exact-extract*.

A naive optimal decoder can calculate the probability of every span $a = T_{i:i+j}$ individually.⁴ Using parallel computation hardware, this would require processing a batch of n^2 spans of up to length n (where $n = |T|$), resulting in $O(n^3)$ space complexity. In contrast, *exact-extract* uses dynamic programming to efficiently perform the same computation, with only $O(n^2)$ complexity.

The *exact-extract* algorithm is based on the observation that every span $a = T_{i:i+j}$ is the j th prefix of the suffix $T_{i:n}$. Thus, for each suffix $T_{i:n}$, we can compute the probability of all of its prefixes $T_{i:i+j}$ in a single decoder forward pass. This process allows to calculate the probability of all possible spans, and select the one with the highest probability, making the algorithm both *extractive* and *exact*.

We now turn to a formal description of the algorithm. Let $\ell(\cdot, \cdot)$ and $e(\cdot, \cdot)$ denote local log-probabilities induced by the model P :⁵

$$\begin{aligned} \ell(i, k) &= \log P(T_{i+k} | T_{i:i+k}) \\ e(i, k) &= \log P(\text{eos} | T_{i:i+k}) \end{aligned}$$

Here, $\ell(i, k)$ is the log-probability of predicting the k -th token in the suffix $T_{i:n}$ (given its prefix $T_{i:i+k}$, à la teacher forcing), while $e(i, k)$ is the log-probability of ending the generated sequence at this point.

For a fixed i , both $\ell(i, \cdot)$ and $e(i, \cdot)$ are calculated in a single decoder forward pass, as we simply "pool" the log-probability of the next token and the eos token for each prediction. We therefore need only n decoder passes in order to derive $\ell(i, j)$ and $e(i, j)$ for all i, j .

Next, we denote the *cumulative* log-probability of a sequence using $L(\cdot, \cdot)$:

$$\begin{aligned} L(i, j) &= \log P(T_{i:i+j}) \\ &= \ell(i, 0) + \dots + \ell(i, j - 1) \end{aligned}$$

2. An *extractive* algorithm is one that can only generate a span from the given input passage T .
 3. A decoding algorithm is considered *exact* if it always generates the most likely sequence, as defined by the underlying model P , i.e. $\arg \max_a P(a|T, Q)$.
 4. We use Python-style span notations, i.e. zero-based indexing and exclusive boundaries. For example, $T_{2:4}$ refers to the span containing the third (T_2) and fourth (T_3) tokens.
 5. For clarity of notation, we assume that P is always conditioned on T and Q as well, i.e. $P(x) = P(x|T, Q)$.

| Dataset | Decoding Algorithm | #Examples | | | | | | | | |
|------------|----------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | | 0 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | All |
| SQuAD | <i>Greedy</i> | 50.4 | 81.3 | 84.1 | 86.0 | 88.3 | 89.0 | 90.3 | 91.2 | 94.5 |
| | <i>Exact-Extract</i> | 60.0 | 82.6 | 85.2 | 86.7 | 89.0 | 89.5 | 90.5 | 91.2 | 94.4 |
| TriviaQA | <i>Greedy</i> | 61.7 | 70.6 | 67.8 | 67.7 | 70.5 | 73.4 | 76.7 | 79.9 | 82.8 |
| | <i>Exact-Extract</i> | 67.9 | 74.8 | 74.8 | 75.3 | 76.7 | 77.6 | 79.0 | 80.5 | 83.4 |
| NaturalQs | <i>Greedy</i> | 42.1 | 61.4 | 63.8 | 65.5 | 67.8 | 69.6 | 71.2 | 72.4 | 81.0 |
| | <i>Exact-Extract</i> | 55.4 | 64.4 | 66.7 | 68.5 | 69.9 | 71.2 | 72.9 | 73.6 | 81.7 |
| NewsQA | <i>Greedy</i> | 19.2 | 41.7 | 45.3 | 45.3 | 48.0 | 51.6 | 56.3 | 61.4 | 71.0 |
| | <i>Exact-Extract</i> | 36.3 | 44.7 | 48.8 | 49.9 | 51.8 | 55.2 | 58.3 | 62.3 | 71.8 |
| SearchQA | <i>Greedy</i> | 24.0 | 61.9 | 61.8 | 69.4 | 71.3 | 77.7 | 80.4 | 83.0 | 87.8 |
| | <i>Exact-Extract</i> | 34.7 | 64.1 | 66.2 | 71.7 | 73.4 | 78.9 | 80.8 | 82.9 | 87.6 |
| HotpotQA | <i>Greedy</i> | 43.3 | 66.3 | 70.3 | 73.1 | 74.6 | 76.4 | 77.4 | 78.7 | 83.0 |
| | <i>Exact-Extract</i> | 51.3 | 65.9 | 69.7 | 72.7 | 74.3 | 75.9 | 76.8 | 78.3 | 82.1 |
| BioASQ | <i>Greedy</i> | 55.5 | 74.7 | 76.8 | 80.4 | 85.2 | 89.9 | 92.2 | 94.2 | – |
| | <i>Exact-Extract</i> | 62.8 | 73.8 | 76.4 | 80.1 | 83.9 | 88.9 | 91.3 | 93.3 | – |
| TextbookQA | <i>Greedy</i> | 17.8 | 41.6 | 42.6 | 47.5 | 52.3 | 60.0 | 70.0 | 73.5 | – |
| | <i>Exact-Extract</i> | 36.0 | 49.9 | 51.2 | 55.6 | 58.0 | 62.6 | 70.8 | 73.4 | – |

Table 1: Performance (F1) across all datasets and training set sizes of the few-shot QA benchmark, as well as the zero-shot setting (**0** examples, no fine-tuning), and the full-data setting (**all** examples) as in the 2019 MRQA Shared Task, containing an order of 100,000 training examples per dataset.

In exact-extract, this value is dynamically calculated using a recursive formula:

$$L(i, 0) = 0$$

$$L(i, j) = L(i, j - 1) + \ell(i, j - 1)$$

At this point, $L(i, j)$ does not take into account the probability of generating the `eos` token. To derive the span’s probability of being the answer, we sum the corresponding cumulative log-probability $L(\cdot, \cdot)$ with that of ending the sequence with an `eos` token $e(\cdot, \cdot)$:

$$\begin{aligned} \log P(a = T_{i:i+j}) &= \log P(T_{i:i+j}) + \log P(\text{eos} | T_{i:i+j}) \\ &= L(i, j) + e(i, j) \end{aligned}$$

Once we calculate $L(i, j)$ and $e(i, j)$ for all (i, j) , we retrieve the most probable span $T_{i:i+j}$ via:

$$i, j = \arg \max_{i, j} (L(i, j) + e(i, j))$$

Note that $L(\cdot, \cdot)$ is calculated directly from $\ell(\cdot, \cdot)$, and together with $e(\cdot, \cdot)$, we can derive the log-probability for all possible spans. Thus, n decoder passes are sufficient for exact-extract, instead of n^2 passes required by the naive optimal decoder.

4. Experimental Setup

To measure how far from optimal is greedy decoding in practice, we compare the performance of exact-extract and greedy decoding on a comprehensive few-shot QA benchmark.

Model We use T5-v1.1 [Raffel et al., 2020, Roberts et al., 2020], an encoder-decoder transformer model pretrained to generate multiple randomly-masked spans.

We choose the v1.1 model checkpoint to avoid data contamination, as it was trained without any labeled data, while the original T5 models were trained in a multitask setting. Our main experiments use T5-large (800M parameters). To corroborate our findings are consistent across model sizes, we also measure the performance of T5-base (250M parameters).

Prompt Following the recent introduction of prompts for few-shot learning [Schick and Schütze, 2021a,b, Gao et al., 2021, Le Scao and Rush, 2021], we align the task of extractive QA with T5’s pretraining objective using a prompt. Specifically, the input to the encoder is:

Text: T
 Question: Q
 Answer:<extra_id_0>.

The model is trained to output:

<extra_id_0> a <extra_id_1>

Here, T and Q are the given passage and question, and a is the expected answer. This specific prompt was selected from 6 candidate prompts as part of the hyperparameter tuning process (see below), as recommended by Perez et al. [2021].⁶

Datasets We report results on the few-shot QA benchmark [Ram et al., 2021], created by subsampling 8 datasets from the MRQA 2019 shared task [Fisch et al., 2019]: SQuAD [Rajpurkar et al., 2016], NewsQA [Trischler et al., 2017], TriviaQA [Joshi et al., 2017], SearchQA [Dunn et al., 2017], HotpotQA [Yang et al., 2018], Natural Questions [Kwiatkowski et al., 2019], BioASQ [Tsatsaronis et al., 2015], and TextbookQA [Kembhavi et al., 2017]. Each dataset has a single fixed test set, and seven different training set sizes on a logarithmic scale from 16 to 1024 examples. To account for sampling variation, five different training sets are sampled for each training set size, accumulating in 35 training sets for each of the 8 datasets. For each dataset, we also examine the zero-shot setting (0 training examples) and the full-data setting (training on all examples). For BioASQ and TextbookQA, the largest setting we examine is 1024 examples, similar to Ram et al. [2021]. Performance is measured via token-level F1 [Rajpurkar et al., 2016] and averaged across the samples of each training set size.

Hyperparameters Hyperparameter tuning can be challenging in a few-shot setting because the development set (which needs to be taken out of an already-small training set) might have insufficient statistical power. To address this issue, we assume that one “academic” dataset is available, which can provide enough validation examples for a modest hyperparameter search. The best hyperparameter configuration found via this single validation set is then used across all datasets and training sizes in our experiments. The academic dataset assumption follows the common practice of reusing hyperparameters tuned on larger data in prior work.

6. See Appendix A for the full set of prompts.

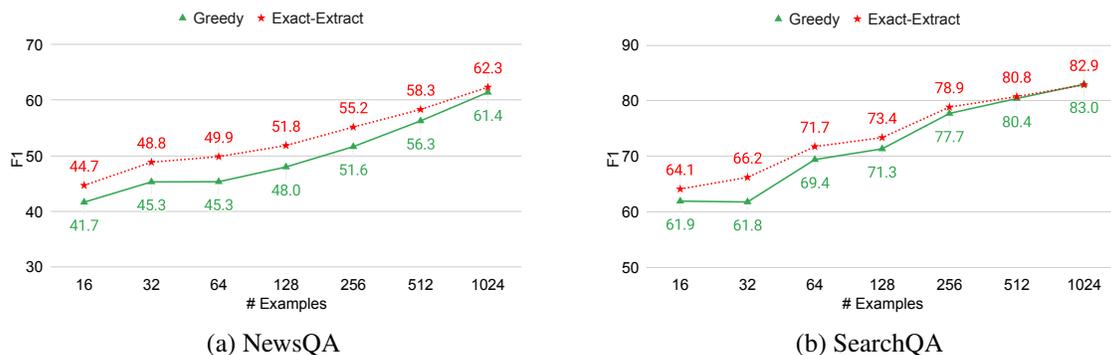


Figure 2: Few-shot Performance (F1) of greedy decoding and exact-extract on NewsQA and SearchQA.

Specifically, we designate SQuAD as our academic dataset for hyperparameter tuning, and sample 2048 examples from its original training set to create a validation set. We ensure that no example in the validation set contains a passage that appears in any of our few-shot training sets. We then apply grid search on the following hyperparameters, for all 35 of SQuAD’s few-shot training sets: learning rate (1e-3, 2e-4, 1e-4, 5e-5), training steps (32, 64, 128, 256, 512, 1024, 2048), and prompts (see all 6 candidates in Appendix A). We select the single hyperparameter setting that optimizes performance across all training set sizes, as described in Appendix B. This process yielded a learning rate of 5e-5 and 512 training steps, as well as the prompt described above. Besides the tuned hyperparameters, we use the Adafactor optimizer [Shazeer and Stern, 2018], a fixed batch size of 32,⁷ and a dropout rate of 0.1. This hyperparameter setting was applied universally to every dataset and data size in our experiments.

We did not perform any additional hyperparameter search for the full training set setups. Instead, we use the same hyperparameters selected for the few shot setting. A single exception is the number of epochs, which is set to 3 for all datasets.

5. Results

We first compare the performance of greedy decoding and exact-extract on the few-shot QA benchmark [Ram et al., 2021]. We observe the gap in performance consistently narrows as the training set get larger. When using 1024 training examples per dataset, greedy decoding lags only 0.3 points behind exact-extract on average. We then show that greedy decoding becomes more extractive (and even more exact) as the training set increases in size, in line with the narrowing gap in performance.

5.1 Performance

Table 1 shows our main performance results, covering all scenarios from zero-shot learning (0 examples) through few-shot learning (16 to 1024 examples) to the full-data setting (an order of 100,000 examples per dataset). The largest difference in performance is observed in the zero-shot setting, when no training examples are used. There, the advantage of exact-extract over greedy is substantial, with margins ranging from 6.2 points (TriviaQA) up to 18.2 (TextbookQA). The large

⁷ For 16 training examples we use a batch size of 16.

gaps across all datasets in the zero-shot setting suggest that when no task-specific training data is available, enforcing extractiveness and exactness through the decoding algorithm can greatly improve performance.

Nevertheless, when some annotated data *is* available, the gap between greedy decoding and exact-extract shrinks at a dramatic pace. Figure 2 visualizes how increasing the training set closes the gap between the two decoding algorithms on NewsQA and SearchQA. We observe that even 16 examples are sufficient to shrink the large gaps in the zero-shot setting to more modest, single-digit gaps, such as 3.0 points on NewsQA and 2.2 points on SearchQA (compared to 17.1- and 10.7-point gaps in the zero-shot setting, respectively). Besides narrowing the performance gap, the shift from 0 to 16 labeled examples also results in a large *absolute* improvement in performance, for both algorithms; in SQuAD, for instance, 16 examples are enough for the model to surpass the 80-point threshold.

As the number of examples increase and reach 1024 and beyond (the full dataset), we observe that the performance difference between the two decoding algorithms diminishes, with less than one point separating the two, not necessarily in exact-extract’s favor.

These trends are rather consistent across all datasets. One notable anomaly is the small but consistent advantage of greedy decoding in the BioASQ and HotpotQA datasets. These datasets suffer from tokenization artifacts, which are particularly adversarial for exact-extract. We analyze this phenomenon in depth in Appendix 7, and explain how the greedy algorithm’s *lack* of formal constraints can actually make it more robust to such issues.

We repeat our experiment using T5-base to verify that the observed trends are robust with respect to model size. The full results of this experiment are available in Appendix C. Indeed, the main trend – in which the performance difference between exact-extract and greedy decoding diminishes as more training examples become available – emerges for the base model as well.

Finally, we compare greedy decoding with T5 to another extractive (and exact) system: Splinter [Ram et al., 2021]. Splinter is an encoder-only transformer pretrained on heuristically-generated pseudo-questions, and has shown strong results on the few-shot QA benchmark. The comparison to Splinter is problematic due to different model sizes and pretraining corpora, but T5’s overwhelmingly stronger results do provide yet another signal that the generative approach can be competitive, even when the decoding algorithm has no theoretical guarantees. Detailed results are available in Appendix D.

5.2 How Extractive and Exact is Greedy?

In Section 5.1 we observe that exact-exact substantially outperforms greedy decoding when no training examples are available, but that this gap quickly closes as more examples are added. We hypothesize that the model acquires certain biases during fine-tuning, causing greedy decoding to produce more extractive and exact outputs. We test our hypothesis by directly measuring both the *extractiveness* and the *exactness* of greedy decoding across different training set sizes. Table 2 shows the results.

Extractiveness We measure extractiveness as the percentage of examples for which greedy decoding generated a contiguous substring from the given passage.⁸ Table 2 shows a steep increase

8. For the sake of this analysis, we only count generated sequences that contain at least one alphanumeric character. Punctuation-only outputs (e.g. “.”) are counted as not-extracted even though they appear in the context. These are common in the zero-shot setting.

| Dataset | Metric | #Examples | | | | | | | | |
|------------|----------------|-----------|------|------|------|------|------|------|------|------|
| | | 0 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 | All |
| SQuAD | <i>Extract</i> | 33.1 | 87.4 | 86.0 | 89.2 | 92.1 | 92.7 | 93.9 | 95.3 | 99.5 |
| | <i>Exact</i> | 28.7 | 82.0 | 81.5 | 84.4 | 87.2 | 87.6 | 88.7 | 89.8 | 92.2 |
| TriviaQA | <i>Extract</i> | 68.7 | 87.6 | 84.8 | 83.7 | 85.5 | 88.6 | 91.3 | 94.2 | 92.7 |
| | <i>Exact</i> | 65.6 | 84.7 | 82.1 | 80.8 | 82.7 | 85.7 | 88.4 | 91.3 | 89.2 |
| NaturalQs | <i>Extract</i> | 51.5 | 80.3 | 82.4 | 82.5 | 87.2 | 89.2 | 91.6 | 93.8 | 98.5 |
| | <i>Exact</i> | 42.3 | 78.3 | 80.8 | 80.4 | 85.0 | 86.5 | 88.4 | 90.4 | 94.0 |
| NewsQA | <i>Extract</i> | 22.8 | 60.0 | 62.4 | 58.5 | 61.5 | 68.1 | 76.0 | 86.0 | 96.6 |
| | <i>Exact</i> | 21.2 | 55.8 | 58.9 | 54.9 | 57.9 | 64.5 | 70.8 | 79.2 | 91.1 |
| SearchQA | <i>Extract</i> | 44.9 | 83.8 | 79.0 | 83.6 | 84.4 | 86.9 | 90.0 | 92.5 | 90.9 |
| | <i>Exact</i> | 43.6 | 81.6 | 77.0 | 81.4 | 82.4 | 85.0 | 88.1 | 90.5 | 88.1 |
| HotpotQA | <i>Extract</i> | 60.8 | 89.9 | 91.6 | 94.0 | 95.4 | 96.0 | 96.8 | 97.3 | 99.6 |
| | <i>Exact</i> | 52.8 | 84.5 | 86.0 | 88.4 | 89.9 | 90.1 | 90.8 | 91.1 | 92.5 |
| BioASQ | <i>Extract</i> | 48.2 | 89.1 | 89.1 | 88.6 | 89.3 | 90.5 | 92.8 | 93.8 | – |
| | <i>Exact</i> | 43.2 | 85.9 | 85.7 | 85.8 | 86.0 | 87.5 | 89.7 | 91.0 | – |
| TextbookQA | <i>Extract</i> | 26.2 | 70.5 | 67.8 | 71.1 | 72.1 | 76.8 | 79.5 | 82.2 | – |
| | <i>Exact</i> | 21.2 | 65.5 | 63.8 | 67.9 | 68.5 | 72.6 | 75.4 | 77.8 | – |

Table 2: Extractiveness and exactness of greedy decoding for all training set sizes. Extractiveness is the percentage of generated answers appearing in the passage. Exactness is the percentage of generated texts identical to exact-extract output.

in extractiveness when comparing 0 examples to 16. In SQuAD for example, generating without any fine-tuning (zero-shot) results in only 33.1% extractive outputs, whereas 16 training examples are enough to increase extractiveness to 87.4%. Extractiveness continues to increase as more examples are available, reaching nearly 100% when training on the full dataset. Effectively, the model acquires a *copy bias* from training on labeled examples, which highly correlates with the increase in performance observed in (Table 1).

Exactness We measure exactness as the percentage of examples for which greedy decoding produces the same output produced by exact-extract.

Table 2 shows that there is a significant increase in the two algorithms’ agreement rate as we introduce training examples. However, unlike extractiveness, exactness does not reach nearly 100%. One possible explanation is that greedy decoding sometimes generates longer, yet just as correct, sequences in practice (i.e. greedy outputs "the IRA" while exact-extract outputs "IRA"). This is supported by our finding in Appendix 7.2, where we show the model sometimes produces correct answers that differ from the ones annotated.

6. Pretraining Models to Extract

In Section 5.2 we observe a strong correlation between performance and a model’s tendency to generate answers extracted from the context. Can we make a model more extractive via pretraining?

| Model | SQuAD | TriviaQA | NQ | NewsQA | SearchQA | HotpotQA | BioASQ | TBQA |
|----------------------|------------------|------------------|-------------|------------------|-------------|------------------|-------------|-------------|
| <i>Greedy</i> | 50.4 (33) | 61.7 (69) | 42.1 (51) | 19.2 (23) | 24.0 (45) | 43.3 (61) | 55.5 (48) | 17.8 (26) |
| + RSS | 71.4 (61) | 69.3 (92) | 57.2 (85) | 43.2 (78) | 29.7 (74) | 59.0 (90) | 65.5 (80) | 39.0 (72) |
| <i>Exact-Extract</i> | 60.0 | 67.9 | 55.4 | 36.3 | 34.7 | 51.3 | 62.8 | 36.0 |
| + RSS | 69.4 | 67.8 | 58.1 | 41.0 | 35.6 | 57.1 | 66.9 | 42.7 |

Table 3: **Top:** Zero-shot performance and extractiveness (in parentheses) of greedy decoding, with and without the RSS pretraining phase. When no labeled examples are available, RSS pretraining greatly boosts both performance and extractiveness. **Bottom:** Zero-shot performance of exact-extract, with and without the RSS pretraining phase.

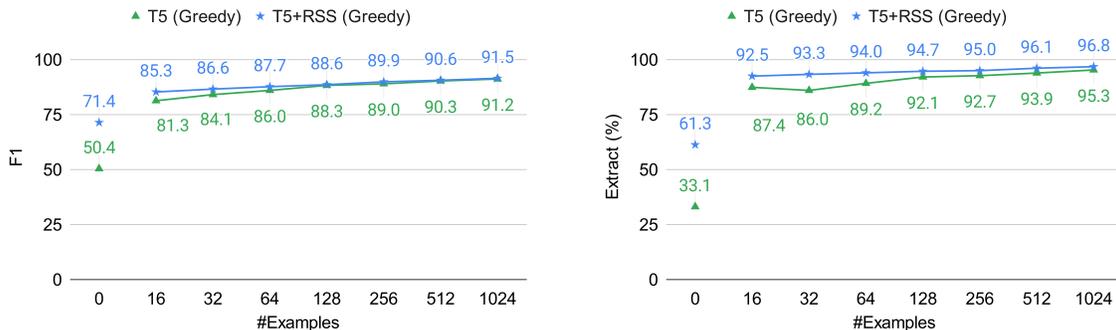


Figure 3: **Left:** Performance of greedy decoding on SQuAD in zero-shot and few-shot settings, with and without the RSS pretraining phase. **Right:** Extractiveness of greedy decoding under the same settings. Performance of exact-extract in these settings is presented in Table 8.

Inspired by recent work on pretraining encoders for span selection, we propose applying an additional pretraining phase (mid-training) to T5 before fine-tuning. We adapt the recurring span selection objective (RSS) used in Splinter [Ram et al., 2021, 2022] to the generative setting: (1) find non-stopword spans that occur more than once in a given passage, (2) mask one instance of a recurring span, (3) train the model to predict the original content of the masked span. While Splinter is trained by masking multiple different spans in parallel, we limit ourselves to a single span in each passage to better approximate the target task. For this experiment, we create 100,000 RSS pretraining examples from English Wikipedia, using WikiExtractor [Attardi, 2015]. We pretrain T5-large on this dataset for 3 epochs.⁹ For simplicity, we use the same hyperparameter configuration from Section 4.

Table 3 shows that incorporating RSS pretraining substantially boosts the extractiveness of greedy decoding in the zero-shot setting, as well as its performance. Exact-extract also benefits from RSS pretraining (but the relative performance gains are smaller), even though it is already 100% extractive. Therefore, we hypothesize that RSS pretraining encourages additional properties that benefit extractive question answering, beyond just copying.

That being said, the advantage of adding an RSS pretraining phase wanes as more labeled examples are available, even when greedy decoding is used. Figure 3 shows how the original T5 model quickly catches up on the RSS-pretrained model’s performance on SQuAD. Notably, when

9. The trained model is available via the Transformers library [Wolf et al., 2020]: https://huggingface.co/tau/t5-v1_1-large-rss

| Test Subset | SQuAD | TriviaQA | SearchQA | HotpotQA | BioASQ | TextbookQA |
|-------------|------------|------------|------------|------------|------------|------------|
| S_{out} | 75.2 (3%) | 39.9 (2%) | 72.2 (9%) | 64.2 (6%) | 59.0 (6%) | 46.7 (2%) |
| S_{in} | 91.6 (97%) | 81.2 (98%) | 84.0 (91%) | 79.1 (94%) | 95.6 (94%) | 74.1 (98%) |

Table 4: Performance of exact-extract on two complementary test set subsets: S_{out} and S_{in} . An S_{out} subset contains only examples in which the tokenized answer is *not* a subsequence of the tokenized passage. An S_{in} subset contains the rest of the test set examples. The relative size of each subset appears in parentheses. NaturalQuestions and NewsQA are omitted from this table since their test sets are 100% extractive. Models are the same used to report results on 1024 training examples in Table 1.

using 128 or more labeled examples, the benefit from adding RSS pretraining is less than one F1 point. This behavior is somewhat expected given our observations in Section 5.2, where we observe a steep rise in both extractiveness and performance once annotated examples are introduced. Hence, adding labeled examples might be more consequential than adding an RSS pretraining phase.

7. Error Analysis

In theory, exact-extract is an optimal decoding algorithm. However, the results in Section 5.1 show that greedy decoding sometimes performs better than exact-extract in practice. Analyzing these cases reveals that inconsistent tokenization can cause the annotated answer to become non-extractive, deteriorating the performance of exact-extract (Section 7.1). We then analyze the greedy algorithm’s errors, and observe that almost half the errors are correct answers, even if not always extractive.

7.1 Exact-Extract

In some datasets, such as BioASQ and HotpotQA, we observe that the greedy algorithm performs better on average than exact-extract (see Table 1). A manual analysis reveals that often in these cases the *tokenized* annotated answer is not a subsequence of the *tokenized* passage. For example, a passage containing the text “(1971)” is tokenized as [“_(19”, “71”, “)”], while the answer string “1971” is tokenized as [“_1971”].

To measure the prevalence and effect of this phenomenon, we partition each test set into two: S_{out} and S_{in} . S_{out} subsets include all test examples where the tokenized answer is *not* a subsequence of the tokenized passage. S_{in} subsets include the rest of the test set. Then, for each model from our main experiment (Section 5.1), we measure the performance of exact-extract on S_{out} and S_{in} .

Table 4 shows that exact-extract performs substantially worse on S_{out} subsets. This is expected, as they are designed to contain only answers which cannot be extracted (token-wise) from the passage. In addition, we observe that in the datasets where exact-exact was outperformed by greedy, S_{out} is relatively larger compared to S_{in} .

The tokenization issue behind this phenomenon stems from the way subword token vocabularies are commonly induced [Sennrich et al., 2016, Kudo and Richardson, 2018]. It is quite likely that this phenomenon disappears when using character-level or byte-level tokenization [Shaham and Levy, 2021, Xue et al., 2022]. However, the fact that greedy decoding is *not* 100% extractive actually allows it to overcome tokenization mismatches and generate the annotated answer.

| Category | Frequency |
|-------------------|-----------|
| Incorrect Answer | 51.9% |
| Correct Answer | 48.1% |
| Annotation Error | 17.6% |
| Not Extractive | 30.5% |
| Paraphrase | 23.6% |
| Added Information | 6.9% |

Table 5: Error analysis of greedy decoding, based on models trained on 1024 examples. All cases reflect examples where exact-extract accurately produced the annotated answer, while the greedy algorithm did not.

7.2 Greedy Decoding

We analyze the cases in which exact-extract did produce the annotated answer, but the greedy algorithm did not. This allows us to decouple the model from the decoding algorithm, since we know that the most likely span according to the model is indeed correct. Specifically, we analyzed results from models trained on 1024 examples, sampling up to 20 examples from each dataset.

Table 5 breaks down the errors into a hierarchy of categories, alongside the prevalence of each error type. We observe that approximately half of the errors (48.1%) are semantically correct answers. Of those, about a third account for annotation errors, typically where there can be multiple correct spans but only one appeared in the test set (and the greedy algorithm chose another).

The other two thirds are particularly interesting: they are semantically correct, but on the other hand, they are not extractive. The majority of these cases are paraphrases, where the model elaborates a bit more (annotated: “shamed”, generated: “he shamed him”), or replaces a number-word with the actual number (annotated: “sixty percent”, generated: “60%”). Most curiously, in about a quarter of the correct answers which are not extractive, the model adds information that was not mentioned in the original passage, e.g. generating “Queen Elizabeth II” instead of the span “the Queen”. In contrast with *hallucination*, commonly reported in summarization tasks [Lewis et al., 2020, Zhao et al., 2020], the information added answers is correct.

One can debate whether non-extractive answers are actually correct. On one hand, the task is defined as *extractive* QA. Having said that, these answers do fulfill a potential user’s information need, and may even benefit said user by containing additional context.

8. Conclusions

We investigate the optimality of greedy decoding for extractive question answering by comparing it to *exact-extract*, an optimal decoding algorithm that guarantees both *extractiveness* and *exactness*. While the greedy algorithm lags behind exact-extract in the zero-shot setting, training the model on as few as 16 labeled examples shrinks the performance gap substantially. This gap continues to narrow as more examples are available, typically converging to less than 1 point (F1) when training on 1024 examples. Overall, our results showcase the impressive ability of pretrained language models to adapt to extractive question answering while relying only on a naive decoding algorithm.

Acknowledgements

This work was supported by the Tel Aviv University Data Science Center, Len Blavatnik and the Blavatnik Family foundation, the Alon Scholarship, Intel Corporation, and the Yandex Initiative for Machine Learning.

References

- Giuseppe Attardi. WikiExtractor, 2015. URL <https://github.com/attardi/wikiextractor>.
- 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. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. URL <https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf>.
- Rakesh Chada and Pradeep Natarajan. FewshotQA: A simple framework for few-shot learning of question answering tasks using pre-trained text-to-text models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6081–6090, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.491. URL <https://aclanthology.org/2021.emnlp-main.491>.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL <https://aclanthology.org/N19-1423>.
- Matthew Dunn, Levent Sagun, Mike Higgins, V. Ugur Guney, Volkan Cirik, and Kyunghyun Cho. SearchQA: A new Q&A dataset augmented with context from a search engine, 2017.
- Adam Fisch, Alon Talmor, Robin Jia, Minjoon Seo, Eunsol Choi, and Danqi Chen. MRQA 2019 shared task: Evaluating generalization in reading comprehension. In *Proceedings of the 2nd Workshop on Machine Reading for Question Answering*, pages 1–13, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5801. URL <https://aclanthology.org/D19-5801>.
- Tianyu Gao, Adam Fisch, and Danqi Chen. Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online, August 2021. Association for Computational Linguistics. doi:

- 10.18653/v1/2021.acl-long.295. URL <https://aclanthology.org/2021.acl-long.295>.
- Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880, Online, April 2021. Association for Computational Linguistics. URL <https://aclanthology.org/2021.eacl-main.74>.
- Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL <https://aclanthology.org/P17-1147>.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL <https://aclanthology.org/2020.emnlp-main.550>.
- Aniruddha Kembhavi, Minjoon Seo, Dustin Schwenk, Jonghyun Choi, Ali Farhadi, and Hannaneh Hajishirzi. Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL <https://aclanthology.org/D18-2012>.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, March 2019. doi: 10.1162/tacl_a_00276. URL <https://aclanthology.org/Q19-1026>.
- Teven Le Scao and Alexander Rush. How many data points is a prompt worth? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2627–2636, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.208. URL <https://aclanthology.org/2021.naacl-main.208>.
- Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural*

- Language Learning (CoNLL 2017)*, pages 333–342, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1034. URL <https://aclanthology.org/K17-1034>.
- Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL <https://aclanthology.org/2020.acl-main.703>.
- Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. A unified MRC framework for named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5849–5859, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.519. URL <https://aclanthology.org/2020.acl-main.519>.
- Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *ArXiv*, abs/2105.11447, 2021.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL <http://jmlr.org/papers/v21/20-074.html>.
- Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL <https://aclanthology.org/D16-1264>.
- Ori Ram, Yuval Kirstain, Jonathan Berant, Amir Globerson, and Omer Levy. Few-shot question answering by pretraining span selection. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3066–3079, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.239. URL <https://aclanthology.org/2021.acl-long.239>.
- Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. Learning to retrieve passages without supervision. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2687–2700, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.193. URL <https://aclanthology.org/2022.naacl-main.193>.
- Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5418–5426, Online, November 2020.

- Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.437. URL <https://aclanthology.org/2020.emnlp-main.437>.
- Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online, April 2021a. Association for Computational Linguistics. URL <https://aclanthology.org/2021.eacl-main.20>.
- Timo Schick and Hinrich Schütze. It’s not just size that matters: Small language models are also few-shot learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2339–2352, Online, June 2021b. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.185. URL <https://aclanthology.org/2021.naacl-main.185>.
- Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL <https://aclanthology.org/P16-1162>.
- Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.
- Uri Shaham and Omer Levy. Neural machine translation without embeddings. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 181–186, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.17. URL <https://aclanthology.org/2021.naacl-main.17>.
- Noam M. Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *ArXiv*, abs/1804.04235, 2018.
- Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordani, Philip Bachman, and Kaheer Suleman. NewsQA: A machine comprehension dataset. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 191–200, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2623. URL <https://aclanthology.org/W17-2623>.
- George Tsatsaronis, Georgios Balikas, Prodromos Malakasiotis, Ioannis Partalas, Matthias Zschunke, Michael R. Alvers, Dirk Weissenborn, Anastasia Krithara, Sergios Petridis, Dimitris Polychronopoulos, Yannis Almirantis, John Pavlopoulos, Nicolas Baskiotis, Patrick Gallinari, Thierry Artières, Axel-Cyrille Ngonga Ngomo, Norman Heino, Eric Gaussier, Liliana Barrio-Alvers, Michael Schroeder, Ion Androutsopoulos, and Georgios Paliouras. An overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition. *BMC Bioinformatics*, 16(1):138, April 2015. ISSN 1471-2105. doi: 10.1186/s12859-015-0564-6. URL <https://doi.org/10.1186/s12859-015-0564-6>.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL <https://aclanthology.org/2020.emnlp-demos.6>.

Wei Wu, Fei Wang, Arianna Yuan, Fei Wu, and Jiwei Li. CorefQA: Coreference resolution as query-based span prediction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6953–6963, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.622. URL <https://aclanthology.org/2020.acl-main.622>.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL <https://aclanthology.org/2022.tacl-1.17>.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium, October–November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1259. URL <https://aclanthology.org/D18-1259>.

Zheng Zhao, Shay B. Cohen, and Bonnie Webber. Reducing quantity hallucinations in abstractive summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2237–2249, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.203. URL <https://aclanthology.org/2020.findings-emnlp.203>.

Appendix A. Hyperparameter Search Space

Our search space includes three hyperparameters: learning rate, number of training steps and the prompt. We choose from the following candidate sets:

- Learning rates: {1e-3, 2e-4, 1e-4, 5e-5}
- Number of training steps: {32, 64, ..., 2048}
- Prompts: See Table 6 for the list of prompts considered.

Following the hyperparameters selection process (see Appendix B), we proceed with a learning rate of 5e-5 and train for 512 steps, with the second prompt from Table 6.

Appendix B. Hyperparameter Selection

We describe our approach for selecting the best hyperparameter configuration. As described in Section 4, we use SQuAD’s 35 training sets; 7 different sizes with 5 sets each, alongside a 2048-example validation set.

Formally, denote the set of training sizes by $N = \{16, 32, \dots, 1024\}$ and the number of different sets for each size by K ($K = 5$ in our case). We define $s_i^{n,k}$ as the model performance on the validation set when trained on the k -th training set of size $n \in N$, using the hyperparameter configuration h_i .¹⁰ Following, we take s_i^n to be the score of h_i averaged across datasets of size n , i.e:

$$s_i^n = \frac{1}{K} \sum_{k=1}^K s_i^{n,k}$$

Next, we normalize s_i^n by the maximal averaged score on datasets of size n :

$$\tilde{s}_i^n = \frac{s_i^n}{\max_j s_j^n}$$

Finally, we average h_i ’s normalized scores across sizes:

$$s_i = \frac{1}{|N|} \sum_{n \in N} \tilde{s}_i^n$$

The hyperparameters configuration h_{i^*} is chosen via $i^* = \arg \max_i s_i$.

Appendix C. Results with T5-base

Table 7 shows performance results when using T5-base in the zero-shot setting and all few-shot settings. The trends are similar; the gap between exact-extract and greedy decoding narrows as more training examples are present.

Appendix D. Comparison with Splinter

We present T5-large and T5-base greedy decoding results alongside those of Splinter-large¹¹ in Table 9.

Although the models cannot be fairly compared (due to different sizes, training corpora and duration of training), T5-large outperforms Splinter-large across all datasets and size regimes; the margin ranges from 14 F1 points on average for 16-64 examples, to 9 points for 128-1024 training examples.

10. h_i defines a specific learning rate, number of training steps and a prompt (see Appendix A).

11. The results reported in Ram et al. [2021] were obtained using Splinter-base. The authors shared new results with us, obtained with Splinter-large.

| |
|--|
| <p><i>T</i> Question: <i>Q</i> Answer:<extra_id_0>.</p> |
| <p>Text: <i>T</i> Question: <i>Q</i> Answer:<extra_id_0>.</p> |
| <p><i>T</i> <i>Q</i> <extra_id_0>.</p> |
| <p><i>T</i> Answer the following question based on the above text: <i>Q</i> <extra_id_0>.</p> |
| <p>Please read the following paragraph and answer the question at the end: <i>T</i> <i>Q</i> <extra_id_0>.</p> |
| <p>Background: <i>T</i> Q: <i>Q</i> A:<extra_id_0></p> |

Table 6: Prompts considered during hyperparameter grid search. The placeholders *T* and *Q* are replaced with the example’s passage and question, respectively; <extra_id_0> is T5’s sentinel token representing a masked span.

| Dataset | Decoding Algorithm | #Examples | | | | | | | |
|------------|----------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | | 0 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| SQuAD | <i>Greedy</i> | 29.7 | 50.7 | 53.3 | 60.3 | 69.6 | 72.8 | 76.4 | 76.5 |
| | <i>Exact-Extract</i> | 34.6 | 54.9 | 57.9 | 62.7 | 71.0 | 73.5 | 77.0 | 76.8 |
| TriviaQA | <i>Greedy</i> | 54.1 | 37.1 | 29.5 | 38.2 | 52.0 | 51.4 | 67.1 | 68.5 |
| | <i>Exact-Extract</i> | 54.5 | 50.9 | 48.0 | 51.7 | 58.3 | 54.8 | 67.5 | 68.5 |
| NaturalQs | <i>Greedy</i> | 13.9 | 35.1 | 39.7 | 44.1 | 50.0 | 52.1 | 54.3 | 55.2 |
| | <i>Exact-Extract</i> | 34.4 | 42.1 | 44.8 | 49.3 | 53.0 | 54.2 | 55.5 | 56.1 |
| NewsQA | <i>Greedy</i> | 25.0 | 20.7 | 22.3 | 26.2 | 34.3 | 39.6 | 42.4 | 44.1 |
| | <i>Exact-Extract</i> | 27.6 | 28.9 | 30.5 | 32.3 | 36.7 | 40.1 | 41.9 | 43.3 |
| SearchQA | <i>Greedy</i> | 4.8 | 29.5 | 27.6 | 38.1 | 51.7 | 59.7 | 65.2 | 64.3 |
| | <i>Exact-Extract</i> | 13.4 | 37.4 | 37.8 | 42.4 | 53.1 | 59.9 | 65.2 | 64.2 |
| HotpotQA | <i>Greedy</i> | 33.3 | 38.5 | 42.5 | 53.7 | 59.5 | 62.5 | 66.0 | 65.5 |
| | <i>Exact-Extract</i> | 41.2 | 40.9 | 44.8 | 54.5 | 59.6 | 62.3 | 65.5 | 64.6 |
| BioASQ | <i>Greedy</i> | 42.8 | 39.5 | 51.0 | 63.1 | 73.8 | 79.3 | 81.9 | 81.9 |
| | <i>Exact-Extract</i> | 46.5 | 42.3 | 52.0 | 64.2 | 72.9 | 79.4 | 82.1 | 81.9 |
| TextbookQA | <i>Greedy</i> | 9.0 | 8.8 | 9.7 | 14.4 | 21.1 | 34.8 | 43.6 | 48.6 |
| | <i>Exact-Extract</i> | 18.9 | 17.2 | 18.7 | 19.9 | 24.9 | 36.2 | 43.9 | 48.1 |

Table 7: Performance (F1) of T5-base across all datasets and training set sizes of the few-shot QA benchmark, as well as the zero-shot setting (0 examples, no fine-tuning) as in the 2019 MRQA Shared Task.

| Model | #Examples | | | | | | | |
|----------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | 0 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| <i>Greedy</i> | 50.4 | 81.3 | 84.1 | 86.0 | 88.3 | 89.0 | 90.3 | 91.2 |
| +RSS | 71.4 | 85.3 | 86.6 | 87.7 | 88.6 | 89.9 | 90.6 | 91.5 |
| <i>Exact-Extract</i> | 60.0 | 82.6 | 85.2 | 86.7 | 89.0 | 89.5 | 90.5 | 91.2 |
| +RSS | 69.4 | 85.6 | 86.7 | 87.9 | 89.4 | 90.2 | 90.7 | 91.9 |

Table 8: Performance (F1) of T5-large on SQuAD across all training set sizes in few-shot QA benchmark, as well as the zero-shot setting (0 examples, no finetuning on the task). Performance is measured for both greedy decoding and exact-extract, with and without the RSS pretraining phase. Exact-Extract shows very little to no advantage over greedy decoding once sufficient amount of examples is available, both for greedy decoding and exact-extract.

| Dataset | Model | #Examples | | | | | | | |
|------------|-----------------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | | 0 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| SQuAD | <i>T5-large</i> | 50.4 | 81.3 | 84.1 | 86.0 | 88.3 | 89.0 | 90.3 | 91.2 |
| | <i>T5-base</i> | 29.7 | 50.7 | 53.3 | 60.3 | 69.6 | 72.8 | 76.4 | 76.5 |
| | <i>Splinter-large</i> | – | – | 70.0 | 75.8 | 80.4 | 81.9 | 85.1 | 86.3 |
| TriviaQA | <i>T5-large</i> | 61.7 | 70.6 | 67.8 | 67.7 | 70.5 | 73.4 | 76.7 | 79.9 |
| | <i>T5-base</i> | 54.1 | 37.1 | 29.5 | 38.2 | 52.0 | 51.4 | 67.1 | 68.5 |
| | <i>Splinter-large</i> | – | – | 45.3 | 55.3 | 58.1 | 66.1 | 40.8 | 71.0 |
| NaturalQs | <i>T5-large</i> | 42.1 | 61.4 | 63.8 | 65.5 | 67.8 | 69.6 | 71.2 | 72.4 |
| | <i>T5-base</i> | 13.9 | 35.1 | 39.7 | 44.1 | 50.0 | 52.1 | 54.3 | 55.2 |
| | <i>Splinter-large</i> | – | – | 40.6 | 46.3 | 54.4 | 48.8 | 64.1 | 67.9 |
| NewsQA | <i>T5-large</i> | 19.2 | 41.7 | 45.3 | 45.3 | 48.0 | 51.6 | 56.3 | 61.4 |
| | <i>T5-base</i> | 25.0 | 20.7 | 22.3 | 26.2 | 34.3 | 39.6 | 42.4 | 44.1 |
| | <i>Splinter-large</i> | – | – | 33.7 | 36.0 | 47.7 | 52.3 | 57.4 | 58.5 |
| SearchQA | <i>T5-large</i> | 24.0 | 61.9 | 61.8 | 69.4 | 71.3 | 77.7 | 80.4 | 83.0 |
| | <i>T5-base</i> | 4.8 | 29.5 | 27.6 | 38.1 | 51.7 | 59.7 | 65.2 | 64.3 |
| | <i>Splinter-large</i> | – | – | 39.9 | 42.0 | 52.0 | 60.7 | 65.0 | 68.5 |
| HotpotQA | <i>T5-large</i> | 43.3 | 66.3 | 70.3 | 73.1 | 74.6 | 76.4 | 77.4 | 78.7 |
| | <i>T5-base</i> | 33.3 | 38.5 | 42.5 | 53.7 | 59.5 | 62.5 | 66.0 | 65.5 |
| | <i>Splinter-large</i> | – | – | 53.2 | 60.5 | 65.5 | 55.7 | 72.1 | 74.1 |
| BioASQ | <i>T5-large</i> | 55.5 | 74.7 | 76.8 | 80.4 | 85.2 | 89.9 | 92.2 | 94.2 |
| | <i>T5-base</i> | 42.8 | 39.5 | 51.0 | 63.1 | 73.8 | 79.3 | 81.9 | 81.9 |
| | <i>Splinter-large</i> | – | – | 58.8 | 55.1 | 77.0 | 82.3 | 86.7 | 91.4 |
| TextbookQA | <i>T5-large</i> | 17.8 | 41.6 | 42.6 | 47.5 | 52.3 | 60.0 | 70.0 | 73.5 |
| | <i>T5-base</i> | 9.0 | 8.8 | 9.7 | 14.4 | 21.1 | 34.8 | 43.6 | 48.6 |
| | <i>Splinter-large</i> | – | – | 39.5 | 47.7 | 52.2 | 57.5 | 49.7 | 51.6 |

Table 9: Performance (F1) of T5-large (greedy decoding), T5-base (greedy decoding) and Splinter-large [Ram et al., 2021], across all datasets and training set sizes of the few-shot QA benchmark, as well as the zero-shot setting (0 examples, no fine-tuning), and the full-data setting (all examples) as in the 2019 MRQA Shared Task, containing an order of 100,000 training examples per dataset. Splinter-large results were available for 32 examples or more.