Atlas

1. 解决什么问题（动机）

- LLM在知识密集型任务上表现不佳，引入RAG可以缓解该问题，但是现有的方法没有验证这种工作对少样本场景和零样本场景是否有用
  - 先前的工作没有探索RAG在少样本学习场景下对LLM的作用
- work we present ATLAS, a carefully designed and pre-trained retrieval augmented language model able to learn knowledge intensive tasks with very few training examples. We perform

2. 如何解决

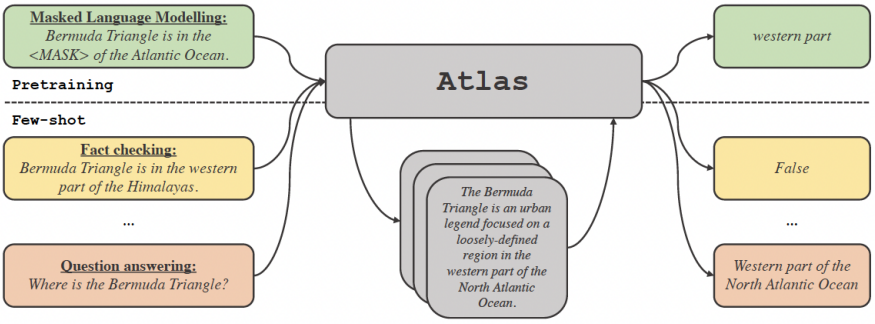

Figure 1: We introduce ATLAS, a retrieval-augmented language model that exhibits strong few-shot performance on knowledge tasks, and uses retrieval during both pre-training and fine-tuning.

- 解决方案：在预训练和微调阶段都引入RAG进行知识增强
- 使用模型：LLM（T5）、检索器（Contriever）
  - T5原论文训练框架

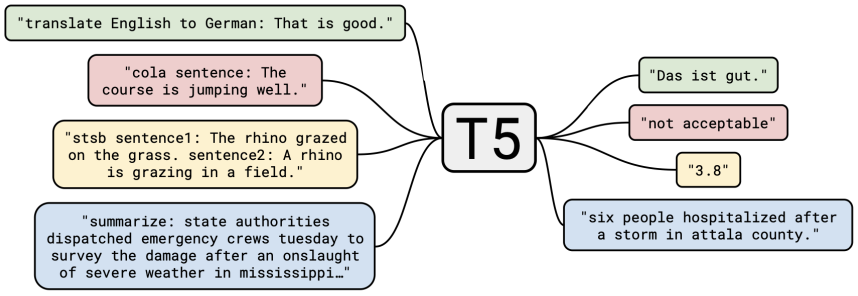

Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".

    - 使用了多种任务来进行训练
  - Contriever原始框架
    - 采用cross encoder结构
    - 使用无监督对比学习来进行训练–如何构造正样本
      - Inverse Cloze Task: The first view is obtained by randomly sampling a span of tokens from a segment of text, while the complement of the span forms the second view
      - Independent cropping: In the context of text, cropping is equivalent to sampling a span of tokens.
      - Additional data augmentation: random word deletion, replacement or masking.
      - 使用MoCo方法来构造负样本

- 召回效果对比

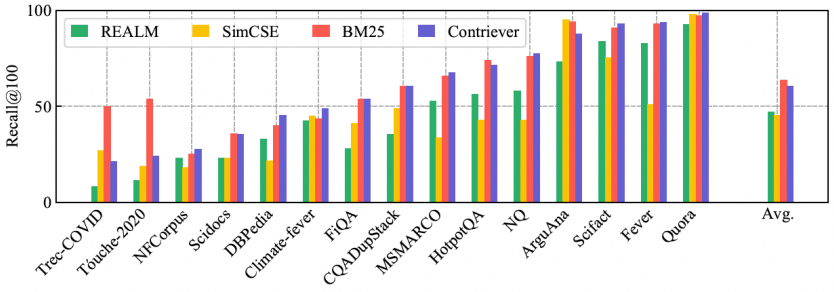

Figure 1: **Unsupervised retrieval.** We compare our pre-training without using *any* annotated data to REALM (Guu et al., 2020), SimCSE (Gao et al., 2021) and BM25. For SimCSE we report results of the model using RoBERTa large. REALM uses annotated entity recognition data for training. We highlight that our unsupervised pre-training is on par with BM25 but on 2 datasets.

- 如何训练
  - We consider loss functions that leverage the language model to provide supervisory signal to train the retriever.
  - 训练检索器
    - ADist

Attention Distillation (ADist). The first loss that we consider is based on the attention scores of the language model, and is heavily inspired by Izacard & Grave (2021). The main idea is that the cross-attention scores between the input documents and the output, can be used as a proxy of the importance of each input document when generating the output. In particular, Izacard & Grave (2021) showed that these scores can be aggregated across attention heads, layers and tokens for a given document to obtain a single score for each document. Then, these scores can be distilled into the retriever by minimizing the KL-divergence with the probability distribution p_{RETR} over the top-K documents $\{\mathbf{d}_k\}_{1,\dots,K}$ obtained from the retriever:

$$p_{\text{RETR}}(\mathbf{d} \mid \mathbf{q}) = \frac{\exp(s(\mathbf{d},\mathbf{q})/\theta)}{\sum_{k=1}^{K}\exp(s(\mathbf{d}_k,\mathbf{q})/\theta)}, \tag{1}$$

where s is the dot-product between the query and documents vectors and θ is a temperature hyper-parameter.

In the original paper, it was proposed to use the pre-softmax scores from the decoder cross-attentions, and average across heads, layers and tokens. Here, we propose an alternative which gives slightly stronger results, which relies on the following observation. In the attention mechanism, as defined by

$$\mathbf{y} = \sum_{n=1}^{N} \alpha_n \mathbf{v}_n,$$

the contribution to the output $\mathbf{y}$ of a particular token n cannot be evaluated from the attention score α_n alone, but should also take the norm of the value $\mathbf{v}_n$ into account. Hence, we use the quantity $\alpha_n \|\mathbf{v}_n\|_2$ as the measure of relevance for token n. Following Izacard & Grave (2021), we average these scores over all attention heads, layers, and tokens to obtain a score for each document. We apply the SOFTMAX operator over the resulting scores, to obtain a distribution $p_{\text{ATTN}}(\mathbf{d}_k)$ over the top-K retrieved documents. We then minimize the KL-divergence between $p_{\text{ATTN}}(\mathbf{d}_k)$, and the distribution p_{RETR} from the retriever defined in Equation 1:

$$\text{KL}(p_{\text{ATTN}} \parallel p_{\text{RETR}}) = \sum_{k=1}^{K} p_{\text{ATTN}}(\mathbf{d}_k) \log\left(\frac{p_{\text{ATTN}}(\mathbf{d}_k)}{p_{\text{RETR}}(\mathbf{d}_k)}\right).$$

Here, this loss is only used to optimize the parameters of the retriever, and not the language model. When using recent deep learning frameworks, this is achieved by applying a STOPGRADIENT operator on p_{ATTN}.

    - 模型注意力层的输出在平均池化之后再经过softmax得到Pattn
    - EMDR

End-to-end training of Multi-Document Reader and Retriever (EMDR^2). Next, we consider the method introduced by Sachan et al. (2021), which is inspired by the expectation-maximization algorithm, treating retrieved documents as latent variables. Given a query $\mathbf{q}$, the corresponding output $\mathbf{a}$ and the set $\mathcal{D}_K$ of top-K retrieved documents with the current retriever, the EMDR^2 loss to train the retriever is

$$\log\left[\sum_{k=1}^{K} p_{\text{LM}}(\mathbf{a} \mid \mathbf{q},\mathbf{d}_k) p_{\text{RETR}}(\mathbf{d}_k \mid \mathbf{q})\right],$$

where p_{RETR} is again the probability over the top-K documents obtained with the retriever, as defined by Equation 1. Again, only the parameters of the retriever are updated by applying a STOPGRADIENT operator

around p_{LM}. One should note that the probability distribution over documents that maximizes this loss function is an indicator of the document corresponding to the highest probability of the output according to the language model. Finally, in practice, the EMDR^2 loss function is applied at the token level, and not at the sequence level.

- PDist

  Perplexity Distillation (PDist). Third, we discuss a simpler loss function which is loosely inspired by the objectives from the attention distillation and EMDR^2 methods (Izacard & Grave, 2021; Sachan et al., 2021). More precisely, we want to train the retriever to predict how much each document would improve the language model perplexity of the output, given the query. To this end, we minimize the KL-divergence between the documents distribution of the retriever (Eqn. 1), and the documents posterior distribution according to the language model, using a uniform prior:

  $$p_k \propto p_{LM}(\mathbf{a} \mid \mathbf{d}_k, \mathbf{q}).$$

  Using the SOFTMAX operator, we have that

  $$p_k = \frac{\exp(\log p_{LM}(\mathbf{a} \mid \mathbf{d}_k, \mathbf{q}))}{\sum_{i=1}^{K} \exp(\log p_{LM}(\mathbf{a} \mid \mathbf{d}_i, \mathbf{q}))}.$$

- LOOP

  Leave-one-out Perplexity Distillation (LOOP). Finally, we propose an objective based on how much *worse* the prediction of the language model gets, when *removing* one of the top-k retrieved documents. To do so, we compute the log probability of the output for each subset of k-1 documents, and use the negative value as relevance score for each document. Following the previous loss function, we use the softmax operator to obtain a probability distribution over documents:

  $$p_{\text{LOOP}}(\mathbf{d}_k) = \frac{\exp(-\log p_{LM}(\mathbf{a} \mid \mathcal{D}_K \setminus \{\mathbf{d}_k\}, \mathbf{q}))}{\sum_{i=1}^{K} \exp(-\log p_{LM}(\mathbf{a} \mid \mathcal{D}_K \setminus \{\mathbf{d}_i\}, \mathbf{q}))}.$$

  As before, we then minimize the KL-divergence between this distribution, and the one obtained with retriever. This loss is more expensive to compute than PDist and EMDR, but, like ADist, employs the language model more closely to the way it is trained i.e. the LM is trained to be conditioned on a set of K documents. For LOOP, the language model is conditioned on $(K-1)$ documents, rather than a single document as in EMDR^2 and PDist.

  For all losses, we can also use a temperature hyper-parameter when computing the target or retriever distributions to control the distribution's peakiness of, which might be important for some tasks or losses. Indeed, for PDist and LOOP, the perplexity of the output may not vary much when conditioning on different documents, especially in the case of long outputs.

- 预设任务：Pretext tasks
  - In this section, we describe pretext tasks that can be used to jointly pre-train the retriever and the language model using only unsupervised data.
  - Prefix language modeling
    - **Prefix language modeling.** First, we consider a standard language modeling task as potential pre-training objective. To cast language modeling in the text-to-text framework, we consider a chunk of N words, and split this chunk in two sub-sequences of equal length $N/2$. Then, the first sub-sequence is used as the query, and the second corresponds to the output. We thus retrieve relevant documents by using the first sub-sequence of $N/2$ tokens, to generate the output.
      - 前N/2 tokens作为query，后N/2 tokens作为输出
  - Masked language modeling
    - **Masked language modeling.** Second, we consider masked language modeling, as formulated by Raffel et al. (2019). Again, starting from a chunk of N words, we sample k spans of average length 3 tokens, leading to a masking ratio of 15%. We then replace each span by a different special token. The model is then trained to generate the masked spans, each span beginning with the special sentinel mask token that was inserted in the input sequence. We retrieve documents using the masked query, but replace the special mask tokens with a mask token supported by the retriever vocabulary.
      - 掩盖词预测
  - Title to section generation
    - **Title to section generation.** Finally, we consider a more abstractive generation task, generating sections from Wikipedia articles, given the article and section title. Here, the query corresponds to the title of the article, together with the title of the section, and the output corresponds to the text of the section. We exclude sections "See also", "References", "Further reading" and "External links".
      - 即根据文章和章节标题生成维基百科文章的章节
- 如何高效微调检索器
  - 更新检索器–导致需要更新embedding和index，如果检索的知识库比较大，会导致较大的开销，这里会分析一些策略，也许可以减少开销
  - Full index update
    - Next, let N be the number of documents in the index, and P_{RETR} be the number of parameters of the retriever. Then, re-computing the full index has a complexity of $N \times P_{\text{RETR}}$. If we refresh the index every R training steps, we obtain the following overhead:

      $$\frac{N \times P_{\text{RETR}}}{4 \times B \times K \times P_{\text{LM}} \times R}.$$

      If we use the BERT-base architecture for our retriever and T5-XL for our language model, we get $\frac{P_{\text{RETR}}}{P_{\text{LM}}} \approx \frac{1}{25}$, lading to the overhead:

      $$\frac{N}{100 \times B \times K \times R}.$$

      If we use an index containing $37M$ documents (the size of our Wikipedia index), train with a batch size of 64 with 20 retrieved documents and refresh the index every 1000 steps, this results in an overhead of $\sim 30\%$.
  - Re–ranking
    -

Re-ranking. A second strategy is to retrieve a larger number of documents L with the retriever, and to re-embed and rerank these documents with the up-to-date retriever, and pass the resulting top-K to the language model. In that case, the overhead of reranking the top-L documents is equal to $B \times L \times P_{\text{RETR}}$. Since we perform this operation at every time step, the overhead is equal to

$$\frac{L \times P_{\text{RETR}}}{4 \times K \times P_{\text{LM}}}.$$

Using the same assumption as before, we finally get that the overhead is of the order of $\frac{L}{100 \times K}$. If we re-rank 10x more documents than what the language model processes (i.e., $L = 10 \times K$), we get an overhead of 10%. However, note that if many updates are performed on the retriever, the index might still need to be fully

- Query-side fine-tuning.

  Query-side fine-tuning. Finally, the last strategy is to decouple the encoding of the queries and documents. In this case, we fix the parameters corresponding to the document encoder, and only train the parameters corresponding to the query encoder. Thus, the embeddings of documents are fixed, and we do not need to refresh the index, and thus there is no computational overhead. As we will see in practice, the impact of fixing the documents encoder varies greatly for different tasks when a large training dataset is available. For most of the few-shot settings that we consider, query-side finetuning does not have large performance impact, and sometimes even slightly improves performance.

- 实验配置
  - 预训练

    Pre-training. For the pre-training, we initialize the retriever module using the unsupervised *Contriever* model, which uses the BERT-base architecture. We initialize the language model with the T5 pre-trained weight. As the original T5 pre-trained model included supervised data in the training set, we use the version 1.1 models which were trained on unlabeled text only. Specifically, we initialize from the `T5-lm-adapt` variants due to their improved stability.

    For the ablation studies performed in Section 4.3 and Section 4.4, we use T5-XL which contains 3B weights. We pre-train all our models for 10,000 iterations, using AdamW with a batch size of 64 and a learning rate of 10^{-4} for the reader and 10^{-5} for the retriever with linear decay and 1,000 warmup steps. We refresh the index every 1,000 steps. This means that the index is recomputed 10 times during the pre-training, leading to an overhead of around 30%, compared to training with a fixed retriever. We set the number of retrieved documents to 20. We detail the hyperparameters used for the training of our final model at the beginning of Section 4.5.

  - 微调

    Fine-tuning. When performing a downstream task, either in a few-shot setting or with a large training set, we employ fine-tuning to adapt our models to these tasks. For the few-shot KILT ablation experiments, we perform a fixed number of fine-tuning iterations, instead of using early-stopping. More precisely, we decided to use 50 iterations for the 64-shot setting and 200 iterations in the 1024-shot setting. In both cases, we use a batch size of 32 examples, a learning rate of 4×10^{-5} with linear decay and 5 warmup steps for both the reader and the retriever.

  - tricks

    Unlabeled datasets. Finally, we discuss the unlabeled text datasets that we use to train our models, which form the retrieval index. First, we consider the Dec. 20, 2021 Wikipedia dump, for which we keep the lists and infoboxes, which are linearized by adding a semi-colon separator between the entries. We split articles by section, and split long sections into passages of equal sizes and containing less than 200 words. This leads to a total of 37M passages, containing 78 words in average. We also use documents from the 2020-10 common crawl dump, preprocessed with the CCNet pipeline (Wenzek et al., 2020). We perform additional document filtering, in a similar fashion to Gopher (Rae et al., 2021). More precisely, we filter documents based on document length, average word length, ratio of alphanumeric characters and number of repeated tokens. This leads to a total of 350M passages. The same passages are used for the index and model pre-training. During pre-training, we ensure the passage we are training on is filtered out from the retrieved documents, to prevent the model from simply retrieving the passage it is de-nosing/generating, and trivially using it to solve the pre-training task.

3. 实验结果

- 不同预设任务对少样本学习的影响

  Table 2: **Pretext task ablation.** We compare different pretext tasks, used to jointly pre-train our models. Examples are randomly sampled from the training set of the KILT version of the dataset. We report the exact match on NaturalQuestions, the F1 score on Wizard of Wikipedia and the accuracy on FEVER.

	64-shot				1024-shot			
	NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Prefix Language Modelling	41.0	14.5	64.9	40.1	**44.7**	17.9	86.0	49.5
Masked Language Modelling	**42.7**	**14.9**	**69.7**	**42.4**	**44.7**	**18.3**	**88.8**	**50.6**
Title-to-section generation	41.1	15.2	66.1	40.8	45.4	17.9	84.6	49.3

- 检索的知识库的使用的区别

  Table 3: **Index content ablation.** In this table, we report results for models where the content of the index was changed between the pre-training and the fine-tuning.

		64-shot				1024-shot			
Index	Training data	NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Wiki	Wiki	**42.7**	14.9	69.7	**42.4**	44.7	18.3	88.8	**50.6**
Wiki	CC	40.9	**15.3**	67.3	41.2	**44.8**	**18.4**	88.1	50.4
CC	Wiki	32.9	14.5	**72.1**	39.8	37.8	17.1	85.8	46.9
CC	CC	38.4	14.9	70.1	41.1	42.0	17.3	**88.9**	49.4

- 检索器微调的实验的影响

Table 4: **Retriever fine-tuning ablation.** Here, we compare different strategies to fine-tune the retriever in a few-shot setting.

	64-shot				1024-shot			
	NQ	WoW	FEVER	Avg.	NQ	WoW	FEVER	Avg.
Standard fine-tuning	44.3	14.9	73.2	44.1	47.0	18.4	89.7	**51.7**
Top-100 re-ranking	44.2	14.6	75.4	**44.7**	**47.1**	**18.7**	88.9	51.6
Query-side fine-tuning	**45.0**	**15.0**	**77.0**	**45.7**	44.9	17.9	**90.2**	51.0
Fixed retriever	36.8	14.5	72.0	41.1	38.0	17.7	89.3	48.3

- 使用不同的方法微调检索器都会带来一定的收益

- 微调数据量/模型规模对实验结果的影响

Table 5: **Performance on MMLU as a function of model size.**

	5-shot			5-shot (multi-task)			Full / Transfer		
	770M	3B	11B	770M	3B	11B	770M	3B	11B
Closed-book T5	29.2	35.7	36.1	26.5	40.0	43.5	42.4	50.4	54.0
ATLAS	38.9	42.3	43.4	42.1	48.7	56.4	56.3	59.9	65.8
Δ	+9.8	+6.6	+7.3	+15.6	+8.7	+12.9	+13.9	+9.5	+11.8

- 横向比较，可以发现随着数据量的增加，模型的效果都明显增加
  - 同规模大小的模型在不同数据量下的实验结果可以表明
- 纵向比较，可以发现增加模型的大小，可以显著提升模型的效果

- 去噪音推理对实验结果的影响

Table 6: **Standard vs de-biased inference for MMLU** These results are reported for ATLAS-11B, using cyclic permutations for de-biasing, which increases inference costs by a factor of 4×.

	Zero-shot	5-shot	5-shot (multi-task)	Full / Transfer
Standard Inference	36.8	43.4	56.4	65.8
De-biased Inference	47.1	47.9	56.6	66.0

- 如何进行去噪音

De-biased Inference As mentioned in the main text, even though our model is finetuned with data that encourages a uniform prior over answer letters (by permuting which answer option letter is used with which lexical answer option text in training data), this may not be enough to ensure the model has no residual bias towards specific letters. Consider answers a, questions q and a nuisance variable $z \in \mathcal{Z}$, which represents the ordering of the answer options or, equivalently, which answer letter gets assigned to which answer option text. There are 4 answer options in MMLU, and thus $|\mathcal{Z}| = 24$ unique ways they can be ordered, or assigned to given letters. Running our model with our standard inference for a question q, corresponds to calculating $p(a|q = \mathsf{q}, z = \mathsf{z})$ for the answer ordering z that happens to appear in the dataset. We can control for z by running the model with all possible answer orderings in the input, and marginalizing: $p(a|q = \mathsf{q}) = \sum_{z' \in \mathcal{Z}} p(a|q = \mathsf{q}, z = z')p(z = z'|q = \mathsf{q})$, and assuming $p(z = z'|q = \mathsf{q})$ is uniform (no answer ordering is more likely than another), this reduces to simply $p(a|q = \mathsf{q}) \propto \sum_{z' \in \mathcal{Z}} p(a|q = \mathsf{q}, z = z')$. This procedure requires 24 forward passes, one for each answer ordering, so is 24× slower than standard inference. Table 13 shows the result of applying the full permutation de-biasing, which leads to an 12% improvement zero-shot and 6% in 5-shot performance overall. Empirically, using only the cyclic permutations of the answer order provided in the original dataset (of which there are 4) works nearly as well, which is what we report in the main paper, and only increases inference compute by a factor of 4, rather than 24. Cyclic permutation de-biasing improves over standard inference by 10% in zero-shot and 5% in 5-shot. Empirically, de-biased inference is largely unnecessary when training in the 5-shot multitask or full dataset setting, as there is enough data for the model to learn a more uniform prior over the letters.

  - 针对选择题，例如4个选项，随机排列有A（4，4）=24种组合方法，推理的时候会多推23组
- 实验结果表明，该方法主要会提升zero-shot和5-shot场景下的效果

4. 启发（可以借鉴的东西）

- 可以通过改变选项位置来构造数据

5. 参考资料

- 论文：https://arxiv.org/pdf/2208.03299
  - code：https://github.com/facebookresearch/atlas
- T5:https://arxiv.org/pdf/1910.10683
  - code:https://github.com/google-research/text-to-text-transfer-transformer
- Contriever：https://arxiv.org/pdf/2112.09118v4
  - code：https://github.com/facebookresearch/contriever