Long context4 提示压缩2

LLMLingua

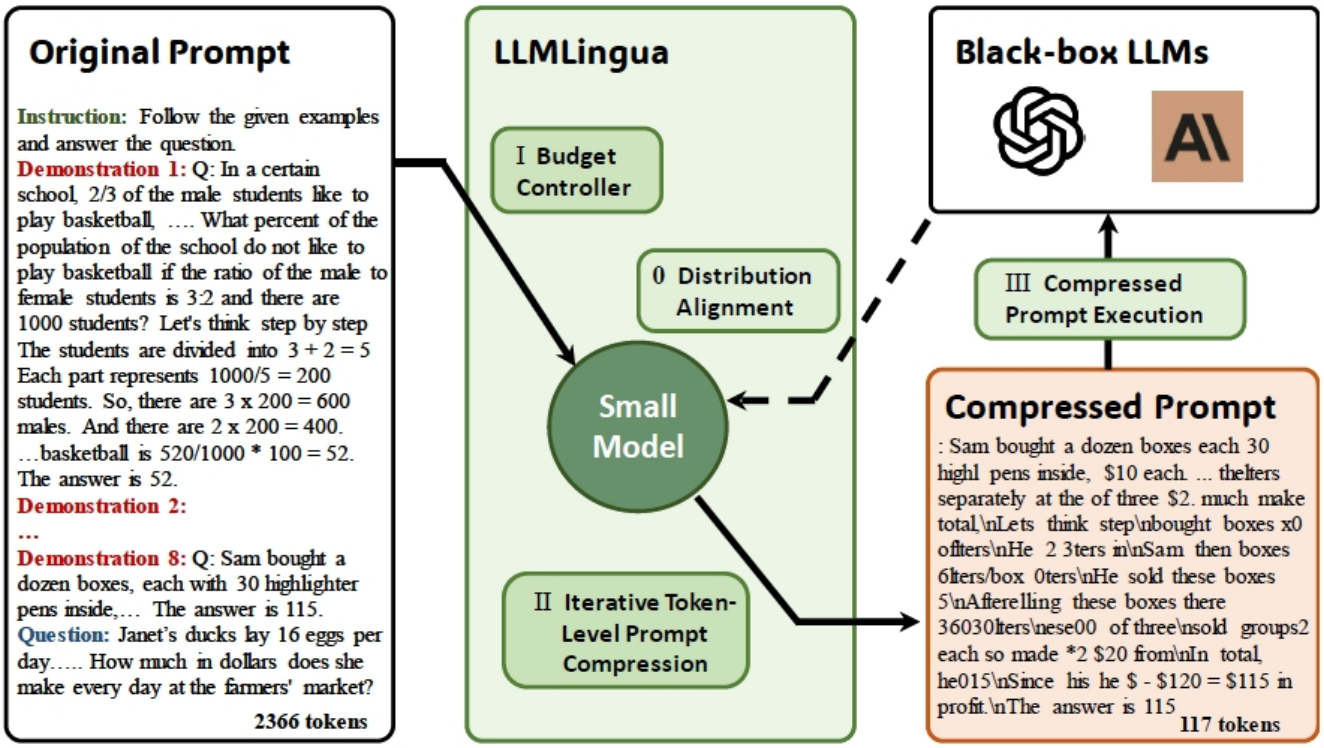


Figure 1: Framework of the proposed approach *LLMLingua*.

LLMLingua 采用预算控制器为原始提示的各个组成部分（例如指令instruction、演示demonstration（其实就是样例）和问题query）动态分配不同的压缩比。$\hat{x}$ 和x分别表示压缩后和压缩前的prompt，$\hat{x_G}$ 和 x_G 分别表示压缩后和压缩前的LLM的输出。训练目标是最小化压缩前输出和压缩后输出分布之间的差距：

$$\min_{\widetilde{\boldsymbol{x}},\tau} \mathrm{KL}(P(\widetilde{\boldsymbol{x}}_G|\widetilde{\boldsymbol{x}}), P(\boldsymbol{x}_G|\boldsymbol{x})), \qquad (1)$$

LLMLingua执行粗粒度、演示级压缩，即使在高压缩比下也能保持语义完整性。此外，LLMLingua 引入了用于细粒度提示压缩的令牌级迭代算法。主要包含三个部分：

- Budget Controller预算控制器：用于demonstration的压缩，从而粗颗粒度地控制budget。一方面过多冗余的demonstration会占据instruction和query的位置，后者对生成答案的影响更大。另一方面token级别的压缩可能导致prompt过于琐碎。因此采用demonstration级别的压缩。

按照perplexity进行排序，只保留perplexity高的demonstration。$\tau = \frac{\hat{L}}{L}$ 表示压缩率，$\hat{L}$ 和L表示压缩后和压缩前的总长度。

$$\tau_{\text{dems}} = \frac{\tau L - (\tau_{\text{ins}} L_{\text{ins}} + \tau_{\text{que}} L_{\text{que}})}{L_{\text{dems}}}. \quad (2)$$

Algorithm 1 Pseudo code of Budget Controller.

Input: A small language model $\mathcal{M}_s$; the original prompt $\boldsymbol{x} = (\boldsymbol{x}^{\text{ins}}, \boldsymbol{x}^{\text{dems}}, \boldsymbol{x}^{\text{que}})$.

1: Set the selected demonstration set $\mathcal{D} = \phi$.
2: Get demonstration compression rate τ_{dem} by Eq.(2).
3: Calculate the perplexity of each demonstration via $\mathcal{M}_s$.
4: Rank all demonstrations in descending order of their perplexity as a list $(\boldsymbol{x}^{\text{dem}}_{(1)}, ..., \boldsymbol{x}^{\text{dem}}_{(N)})$, where N is the number of demonstrations, $\boldsymbol{x}^{\text{dem}}_{(i)}$ is the i-th demonstration.
5: **for** $i = 1$ **do**
6: **if** $\widetilde{L}_{\mathcal{D}} > k \cdot \tau_{\text{dems}} L_{\text{dems}}$ **then**
7: Break.
8: **end if**
9: Append $\boldsymbol{x}^{\text{dem}}_{(i)}$ to $\mathcal{D}$.
10: $i = i + 1$
11: **end for**
12: Allocate remaining budget to $\boldsymbol{x}^{\text{ins}}$ and $\boldsymbol{x}^{\text{que}}$ via Eq. (3).

Output: The subset of demonstrations $\mathcal{D}$ obtained from coarse-grained compression; Additional budget $\Delta\tau_{\text{ins,que}}$ for the instruction and the question.

A small language model, such as GPT-2 or LLaMA

Demonstration-level prompt compression

Allocate the remaining budget to the instruction and the question

$$\Delta\tau = \frac{k \cdot \tau_{\text{dems}} L_{\text{dems}} - \widetilde{L}_{\mathcal{D}}}{L_{\text{ins}} + L_{\text{que}}} \quad (3)$$

- ITPC迭代令牌级提示压缩：对prompt进行进一步的细颗粒度的压缩，得到最终的输出prompt。

粗排按照困惑度进行压缩，依赖于token之间的独立性假设。即在n-gram中提到的，n越小perplexity越高，例如2-gram就是一个token只依赖于前一个token而与其他token无关。换句话说，perplexity高的demonstration，无法把握token之间存在的复杂依赖关系，对于理解语意会造成困难，仅依赖perplexity决定是否保留一个token是不合理的。

ITPC算法在压缩期间更精确地评估每个标记的重要性。它通过迭代处理提示中的每个片段并考虑当前上下文中每个标记的条件概率来实现这一点。这种方法有助于更好地保留令牌之间的依赖关系。

将Budget Controller输出的 $x' = (x^{ins}, x^{D}, x^{que})$ 分成几段 $S = s_1, ..., s_m$ ，用小模型Ms计算每一段的困惑度。每个片段压缩后的拼接到一起，再进行概率估计：

$$\begin{aligned} p(\widetilde{\boldsymbol{s}}_j) &= \prod_{i=1}^{\sum_k^j \widetilde{L}_{s,k}} p(\widetilde{s}_{j,i} | \widetilde{s}_{j,<i}, \widetilde{\boldsymbol{s}}_{<j}) \\ &\approx \prod_{i=1}^{L_{s,j}+\sum_k^{j-1} \widetilde{L}_{s,k}} p(s_{j,i} | s_{j,<i}, \widetilde{\boldsymbol{s}}_{<j}), \end{aligned} \quad (5)$$

where $s_{j,i}$ denotes the i-th token in the j-th segment, $L_{s,j}$ and $\widetilde{L}_{s,j}$ represent the token length of j-th original and compressed segment, respectively.

可以根据每一段的压缩率和PPL分布计算阈值 γ_j ，每一段的压缩率可以总结为：

$$\tau_{s_j} = \begin{cases} \tau_{\text{ins}} + \Delta\tau, & \text{if } s_j \text{ from } x^{\text{ins}}, \\ \tau_{\text{dems}}, & \text{if } s_j \text{ from } x^{\mathcal{D}}, \\ \tau_{\text{que}} + \Delta\tau, & \text{if } s_j \text{ from } x^{\text{que}}. \end{cases} \quad (6)$$

每一段中每个PPL大于阈值 γ_j 的token被保留

$$\widetilde{s}_j = \{s_{j,i} | p(s_{j,i}) > \gamma_j\} \quad (7)$$

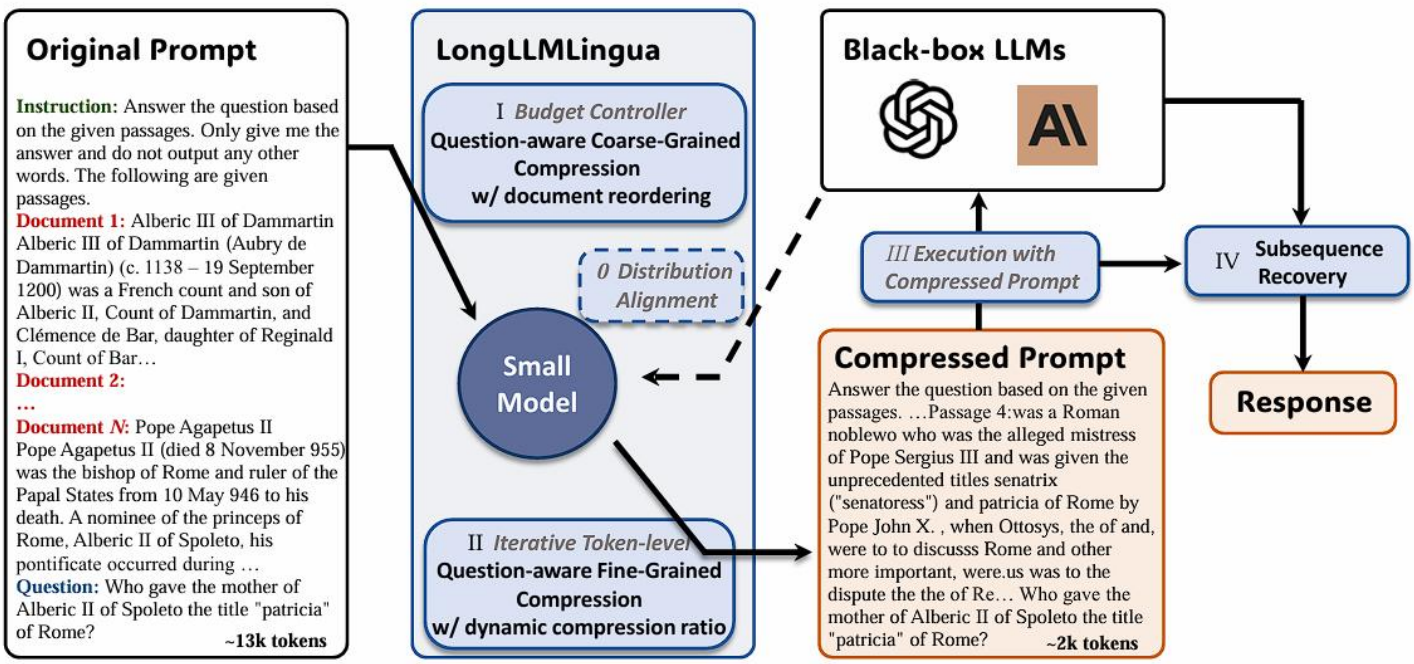
- Distribution Alignment：用于消除压缩用的小模型Ms与LLM之间的分布gap。

利用LLM生成的数据来对Ms进行指令微调，微调目标为：

$$\min_{\theta_s} \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} \mathcal{L}\left(\mathbf{x}_i, \mathbf{y}_{i,\text{LLM}}; \theta_{\mathcal{M}_s}\right)\right], \quad (8)$$

LongLLMLingua

LLMLingua 在压缩过程中没有考虑用户的问题，可能会保留不相关的信息。LongLLMLingua通过将用户问题纳入压缩过程来解决这个问题。



- 问题粗粒度压缩

通过找到一个指标 r_k 来衡量每个document的重要性，并只保留重要性高的document。计算文档级的perplexity $p(x_k^{doc}|x^{que})$ 效果不好，因为文档中包含了大量的无关信息，每个document的PPL值都很高。

因此这篇文章用 $p(x_k^{que}|x^{doc})$ 来衡量PPL，并且在 x^{que} 添加了一句 “We can get the answer to this question in the given documents” 来增强query和document之间的联系，并减轻幻觉。

$$r_k = -\frac{1}{N_c}\sum_{i}^{N_c} \log p(x_i^{\text{que,restrict}} | \mathbf{x}_k^{\text{doc}}), \quad k \in \{1, 2, \cdots, K\}, \tag{2}$$

- 问题细粒度压缩

衡量instruction、query和document中每个token的重要性。x^{ins}和x^{que} 的压缩和LLMLingua的token压缩，document的压缩需要包含更多的question相关的信息。本文使用对比困惑度，也就是由条件question导致的分布偏移：

$$s_i = \text{perplexity}(x_i | x_{<i}) - \text{perplexity}(x_i | x^{\text{que}}, x_{<i}). \tag{3}$$

实验证明高对比困惑度的token与question更相关。

- 文档重新排序

实验结果表明，LLM倾向于使用提示开头和结尾的内容，而忽略中间的内容。因此将粗粒度压缩后的结果按照 r_k 进行排序，按照分数从前到后降序排列

$$(\mathbf{x}^{\text{ins}}, \mathbf{x}_1^{\text{doc}}, \cdots, \mathbf{x}_{K'}^{\text{doc}}, \mathbf{x}^{\text{que}}) \xrightarrow{r_k} (\mathbf{x}^{\text{ins}}, \mathbf{x}_{r1}^{\text{doc}}, \cdots, \mathbf{x}_{rK'}^{\text{doc}}, \mathbf{x}^{\text{que}}) \tag{4}$$

- 动态压缩比

LLMLingua对所有document使用同样的压缩比。LongLLMLingua 使用粗粒度压缩的重要性分数来指导细粒度压缩期间的预算分配。

首先使用 LLMLingua 的预算控制器设置保留文档的初始预算。然后，在细粒度压缩阶段，动态地将压缩预算分配给每个文档。这种分配基于文档重要性得分的排名指数 $I(r_k)$ ，该得分是在粗粒度压缩阶段确定的。

$$\tau_i = \tau_k^{\text{doc}}, \quad \forall x_i \in \mathbf{x}_k^{\text{doc}},$$
$$\tau_k^{\text{doc}} = \max(\min((1 - \frac{2I(r_k)}{K'})\delta\tau + \tau^{\text{doc}}, 1), 0), \tag{5}$$

where i and k is the index of token and document, K' denotes the number of documents, and $\delta\tau$ is a hyper-parameter that controls the overall budget for dynamic allocation.

- 保证关键信息完整

在细粒度压缩过程中，可能会压缩一些关键名词，比如2009被压缩成209，导致生成的答案有问题。本文提出子序列恢复算法

Algorithm 1 Token-level Subsquence Recovery Algorithm

Input: The original prompt $\boldsymbol{x}$; the compressed prompt $\widetilde{\boldsymbol{x}}$; the generation response of LLMs $\boldsymbol{y}$.

Set the final response list $\boldsymbol{y}_{\text{rec}} = \phi$, the left token index of subsquence l to 0.
while $l < \boldsymbol{y}.len()$ **do**
    if Substring $y_l \in \widetilde{\boldsymbol{x}}$ **then**
        Find the longer substring $\widetilde{\boldsymbol{y}}_{\text{key},l} = \{y_l, y_{l+1}, ..., y_r\} \in \widetilde{\boldsymbol{x}}$.
        Find the maximum common shortest subsequence $\boldsymbol{x}_{i,j} = \{x_i, x_{i+1}, ..., x_j\}$ in the original prompt $\boldsymbol{x}$.
        Add the subsequence $\boldsymbol{x}_{i,j} = \{x_i, x_{i+1}, ..., x_j\}$ to the response $\boldsymbol{y}_{\text{rec}}$.
        Set the left index l to $r + 1$.
    else
        Add the token y_l to the response $\boldsymbol{y}_{\text{rec}}$.
        Set the left index l to $l + 1$.
    end if
end while

Output: The final response list $\boldsymbol{y}_{\text{rec}}$.