大语言模型能力增强——自我认知、持续迭代学习

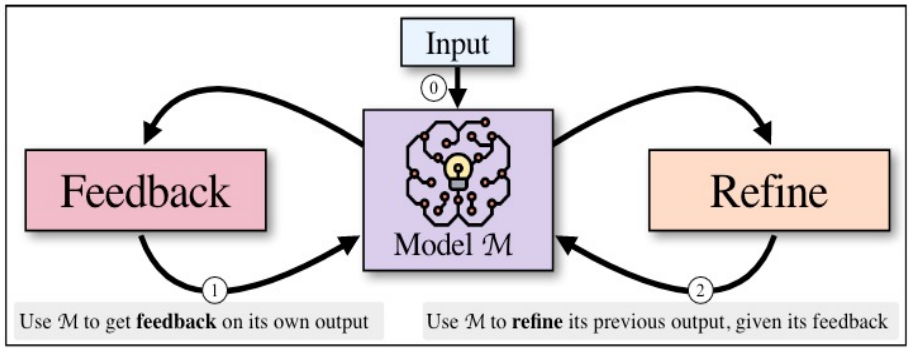
1. 自我认知能力

Without reflection, we go blindly on our way, creating more unintended consequences and failing to achieve anything useful.

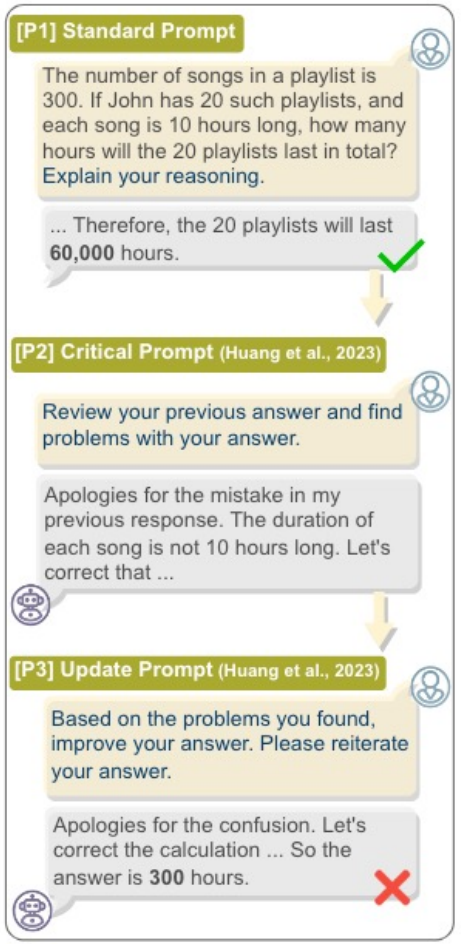
——Margaret J. Wheatley.

小型大语言模型的自我纠错能力

- 为了赋予LLM自我纠错的能力，现有工作设计了一套复杂的pipeline和prompt提示，但仍存在两个问题：
  - 问题1: LLM以显示提示的方式来验证和修改初始答案，而不是像人类一样自发地完成这两个步骤
  - 问题2: 这些prompt engineering的工作所设计的复杂指令，对于开源模型而言很难做到指令跟随



self-correction的一般流程

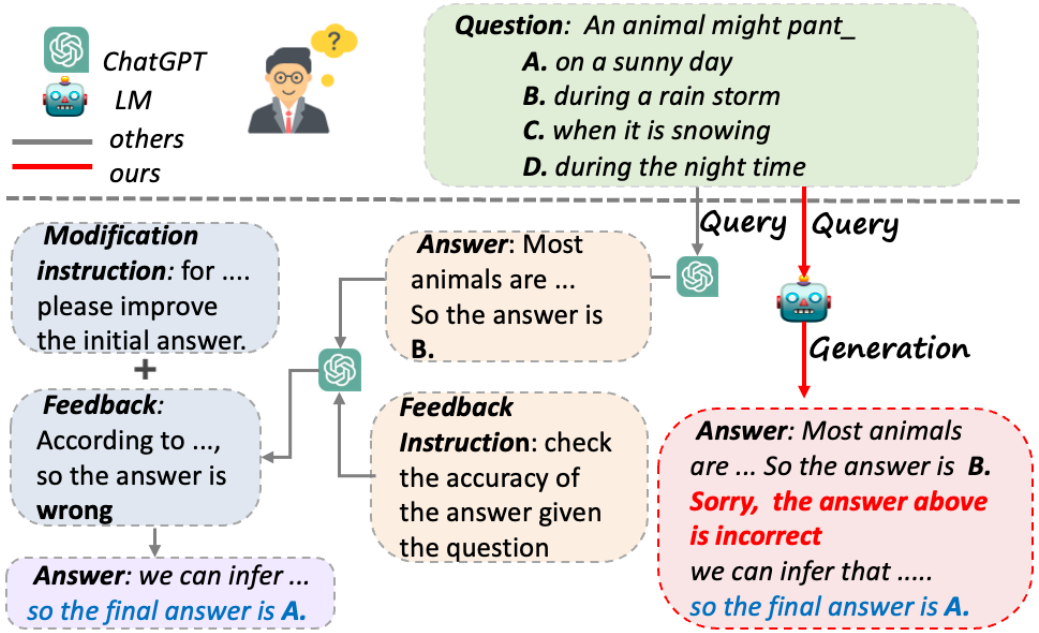


基于Prompt engineering的自我纠错方法

Large language models cannot self-correct reasoning yet [2023]

内在自我纠错方法

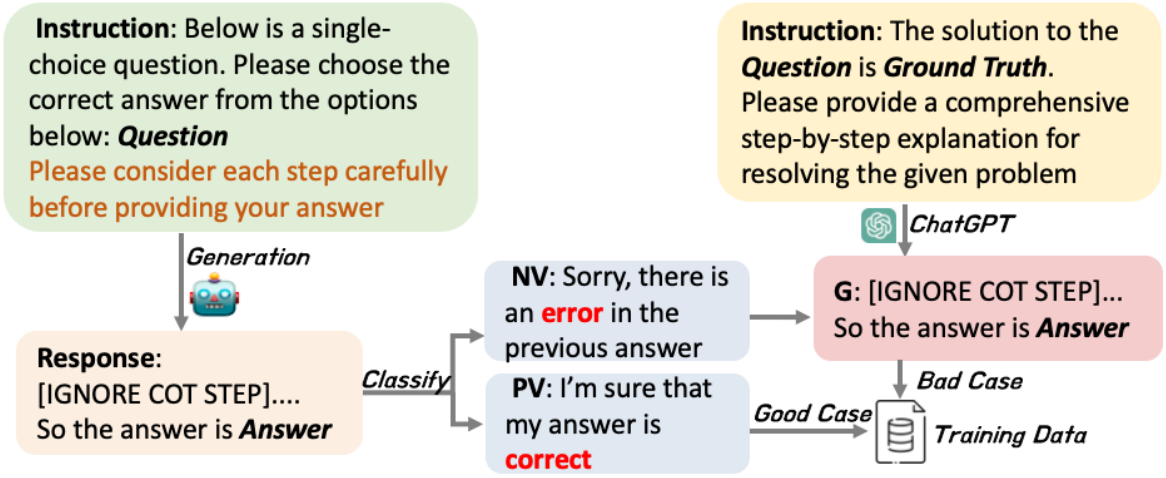
- 为了让小型的大语言模型具备自我纠错的能力，我们提出了Intrinsic Self-Correction (ISC)，这种方法甚至能够适用于6B规模的模型
  - 构建自我纠错指令微调数据集
  - 提出部分掩码技术（Partial Answer Mask）进行指令微调



我们提出的ISC与其他自我纠错方法的区别. 灰色线条部分是prompt engineering的方法，红色部分是我们的方法

自我纠错数据集构建

- 问题准备: 基于公开数据集构造不同任务下的问题
- 答案准备: 使用不同的开源模型进行推理，使用EM来将答案进行分类： good cases, bad cases
- 指令准备：使用两种方式来构建自我纠错的指令
  - 使用ChatGPT进行改写
  - 重新安排不同任务的指令: TaskP, COTP and SCP
- 数据格式: $A_n^1 - COT \parallel A_n^1 \parallel NV \parallel A_n^2 - COT \parallel A_n^2 \cdots A_n^n \parallel$PV



自我纠错数据集构造流程

部分掩码技术（Partial Answer Mask）

- 🙋Good case：训练数据中只有答案部分才用于计算损失
- 🙅Bad case: 不正确的答案不用于计算损失，此时从验证部分计算损失

Good case sample

Question: *(TaskP)* Examine the following options carefully and select the correct one. *(COTP)* Before providing your final answer, give the analysis steps. *(SCP)* And you need double-check your response for accuracy before proceeding to submit.
(question) Where do you buy tickets at a ticket booth for games?
A. train station B. cathedral C. metro station D. fairgrounds E. amusement park
Answer: *(*A_1^1*- COT)* The question mentions the keywords "buying tickets" and "games", so we can guess that this is a · · ·
(A_1^1*)* Therefore, the correct answer is **E. amusement park**
(PV) Thinking about the correctness of the previous answer· · ·
Thinking result: I am sure that the earlier answer is **correct**

Bad case sample

Question: *(TaskP)* Please choose the most appropriate one from the following options:
(question) what contributes more, though less than some believe, to the lung damage caused by smoking?
A. smoking less B. switching to chewing C. no filters D. switching to e-cigs
(COTP) Please give the detailed solving process and *(SCP)* verify your response before final submission.
Answer: *(*A_2^1*-COT)* Smoking causes less lung damage than people think, but it's not completely without effect.
So the answer is **A. smoking less**
(NV) Thinking about the correctness of the previous answer · · ·
Thinking result: Sorry, there is an error in the previous answer.
(SC-COT) Let's analyze each option:
A. smoking less: The question clearly mentions that it contributes less than some people think,· · · · · ·
C. no filters: filters it contributes to lung damage, and to a lesser extent than some believe. Therefore, the no filter option meets the requirement.
(A_2^2*)* So the correct option is **C. no filter**.

实验结果

- Baseline：CuteGPT-7B、CuteGPT-13B、Llama2-7B、ChatGLM-6B、Vicuna-7B、Vicuna-13B
- 我们所提的方法能够使模型在意识到自己生成了错误答案后能够进一步修改
- ChatGLM在OpenBookQA数据集上，ChatgGLM-6B的准确率从37%提升到了42.6%，增益了**5.6%**

Base Models	OpenBookQA		CommonsenseQA	
	ACC-First	ACC	ACC-First	ACC
CuteGPT-7B	25.2	29.0 (+3.8)	23.2	28.9 (+3.7)
CuteGPT-13B	37.2	42.0 (+4.8)	35.6	37.9 (+2.3)
Llama2-7B	52.2	52.2 (+0.0)	52.2	52.3 (+0.1)
ChatGLM-6B	37.0	42.6 (**+5.6**)	34.3	38.7 (**+4.4**)
Vicuna-7B	28.6	28.80 (+0.4)	25.9	26.2 (+0.3)
Vicuna-13B	33.8	34.0 (+0.2)	32.4	32.6 (+0.3)

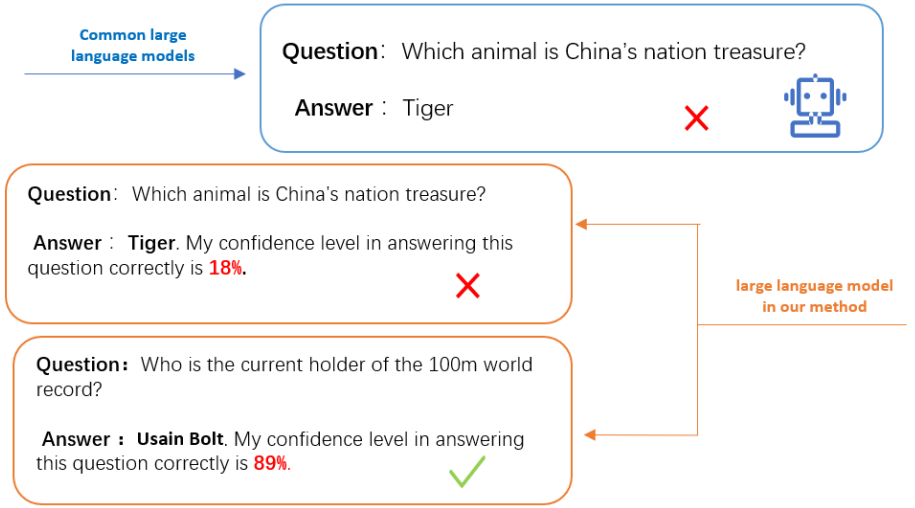
不同模型使用ISC后在两个数据集上的ACC

增强大语言模型的自知之明能力

- 大语言模型在遇到自己不会或者不熟悉的的问题时，仍然以肯定的语气给出错误的答案，导致即使LLM生成错误答案的情况下，用户也无法识别，降低了LLM的可信赖度。

◆如何让模型生成答案的同时给出确定程度？

- 挑战1：few-shot提示的方式无法迁移到开源模型中，很难做到指令跟随
  - 让LLM的生成对于问题的确定程度的工作主要集中在闭源大模型（如ChatGPT和GPT4）
- 挑战2：如何获得LLM对于生成正确答案的确定程度



大模型只给出关于问题的答案。我们关注模型在输出答案的同时，利用自知之明让其提供回答正确问题的概率

增强大语言模型的自知之明能力

✓Step1: 收集大语言模型在问题集上的答题记录

- 采用放回抽样的方式，抽取大量数据（抽5n-10n次，n为问题集题目量）让LLM作答，记录模型在这些问题上的答题情况。

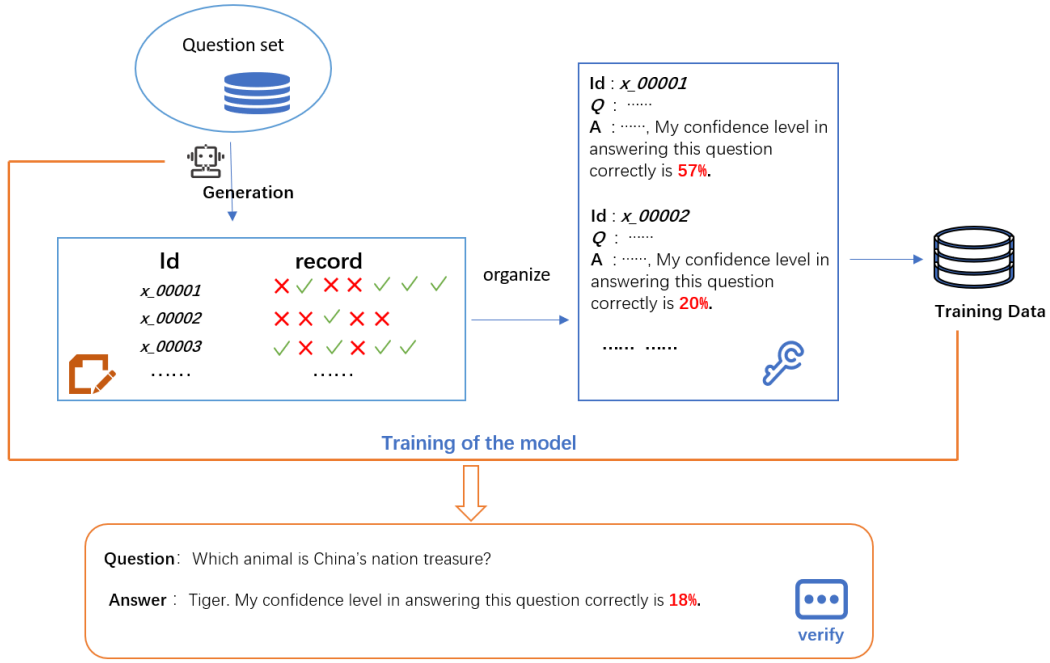
✓Step2: 构造带有回答置信度信息的生成答案

- 根据答题记录统计LLM在训练数据集上的做题水平，用 (题目x的正确次数/题目x的作答次数) 作为模型对该题目的掌握程度，将其作为训练数据的置信度。

✓Step3：构建自知之明训练数据格式

<Question, Answer+Confidence>

✓Step4: 进行指令微调训练



我们提出的增强大模型自知之明能力的流程图

实验结果

- 模型生成的的置信度分数与实际正确率成正相关度极高
- 在XieZhi数据集上，当模型给出置信度在80-100% 的情况下，其实际正确率达到了**89.19%**
- 如果只采纳置信度高于既定阈值的生成结果，那么模型生成正确答案的概率能大幅提升

相关度系数

Dataset	Method	Confidence interval					PCCs
		0-20%	20-40%	40-60%	60-80%	80-100%	
CuteGPT-13B							
C-Eval	Base	34.70%	33.64%	34.16%	32.58%	34.75%	-0.169
	SaKet	32.90%	27.63%	43.76%	47.62%	**50.37%**	**0.838**
XieZhi	Base	44.57%	37.85%	44.59%	49.38%	43.75%	0.381
	SaKet	28.20%	42.71%	48.38%	66.21%	**89.19%**	**0.979**
LLaMA2-Chat-13B							
Gsm8k	Base	35.04%	29.17%	34.28%	36.13%	30.78%	-0.083
	SaKet	23.78%	34.86%	31.91%	48.97%	**54.57%**	**0.879**
CommonsenseQA	Base	55.72%	58.35%	57.03%	55.31%	58.26%	0.230
	SaKet	46.85%	55.23%	69.42%	75.34%	**86.60%**	**0.984**

Table 2: The quantitative results on various datasets

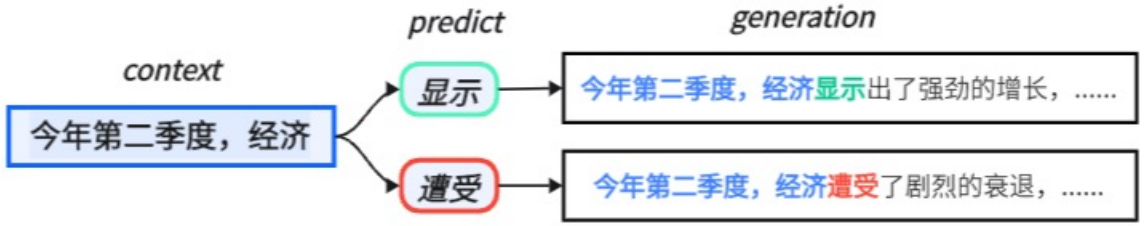
LLM生成的不同置信度分数与其正确率的情况

model	CT	dataset	ACC	ACC_UP	DP
CuteGPT	65%	C-Eval	33.76%	48.90%	21.05%
		XieZhi	44.76%	73.62%	26.33%
LLaMA2-Chat	55%	Gsm8k	30.67%	46.3%	24.30%
		CommonsenseQA	60.20%	77.41%	38.42%

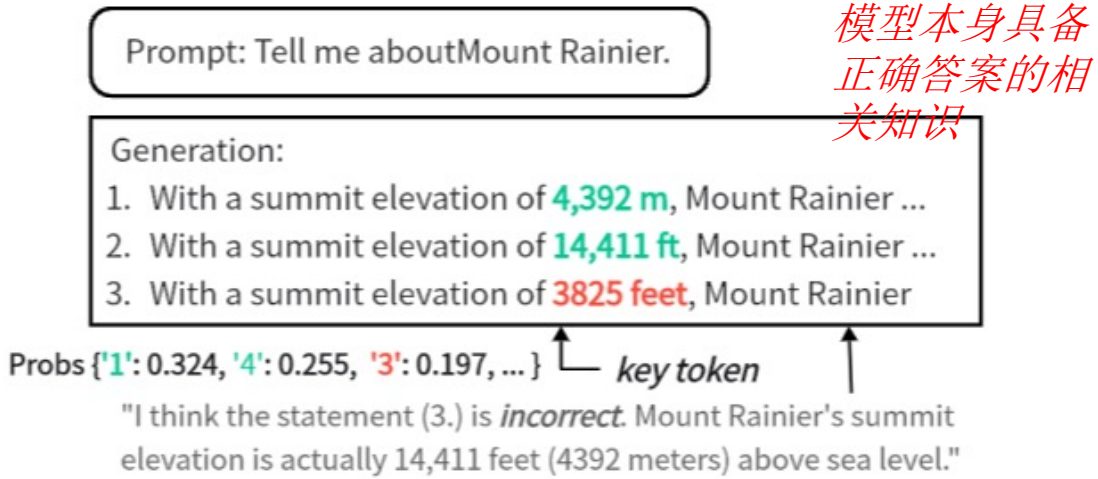
LLM提供的在不同置信度分数下，其生成答案的正确率

基于自我评估的回溯解码

- 目前大模型仍以单向自回归解码为主要的文本生成方式。对于当前位置的解码，不同token选择会显著影响模型后续的文本生成结果。
- 某些关键位置的token选择具有较高的不确定性。若选择不合适则会导致后续生成不正确的文本
- 导致“幻觉”出现的原因并非LLM缺乏回答正确信息的相关知识，可能是模型的解码过程中缺乏识别错误和纠错手段，导致解码过程中的错误累积；



模型对不同词语的选择导致后续生成完全不同的文本



模型本身具备正确答案的相关知识

不同的数字选择会最终影响生成结果的正确性

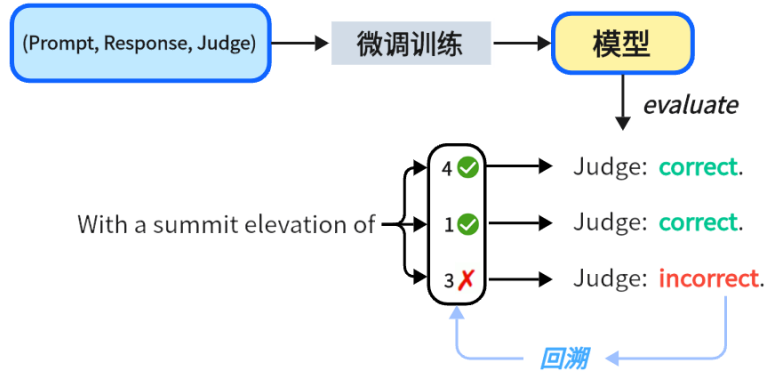
基于自我评估的回溯解码

- 提出基于大模型自我评估的回溯解码策略，让模型在关键位置处选择合适的token来增强生成质量

➢ **增强大模型自我评估的能力：** 构建 <question, response+judge> 的微调数据并进行指令微调训练

➢ **回溯解码：**

- 解码时，在不确定性较高的位置（即关键位置）处重新预测，取概率值最高的前k个token继续生成，并在最后生成每一条推理路径上自我评估的结果
- 选择评估结果最佳的token作为当前token的预测结果，继续生成后续token



回溯解码的过程

2.持续迭代学习能力

The Capacity to Learn is a Gift

— Brian Herbert

从错误中持续迭代进化

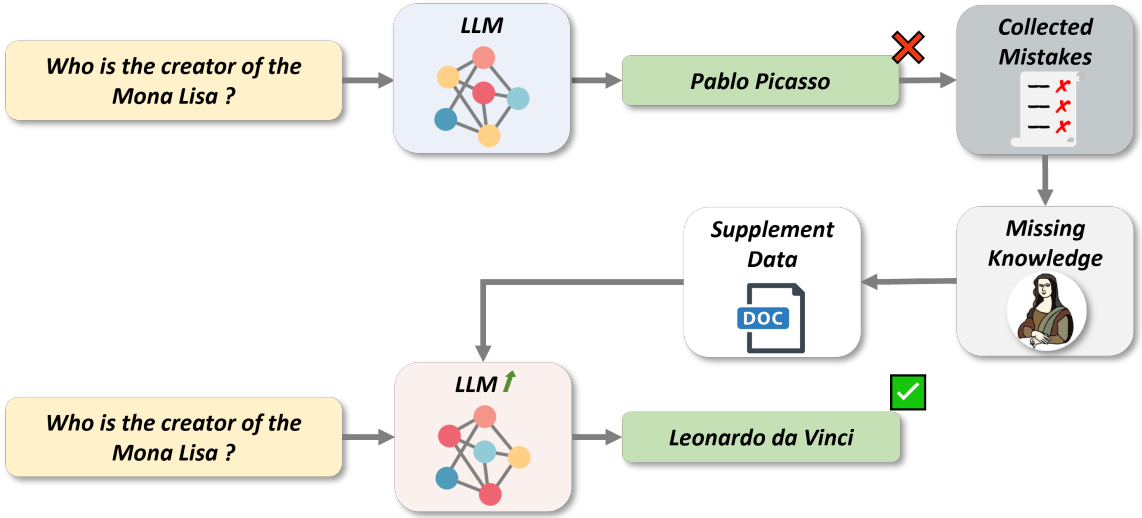
- 让大模型生成正确且令人满意的答案并非是一蹴而就的，仍然会出现“幻觉”。大模型产生幻觉的可能原因有：
  - 问题超过LLM的“知识边界”
  - 参数中的知识或者信息过时
- 需要衡量大模型在任务中的表现针对性地持续优化，从而使得大模型能够不断迭代进化
- 大模型中的持续学习是指：在每一次模型迭代进化的过程中，挖掘和发现大模型在哪些方面表现得较差，从而在接下来更新的版本中能够有效克服这些局限
- 传统持续学习
  - 任务导向
  - 定期使用最新语料库从头训练（成本高昂）
  - 小样本补充训练（灾难性遗忘）

从错误中持续迭代进化（CEM）

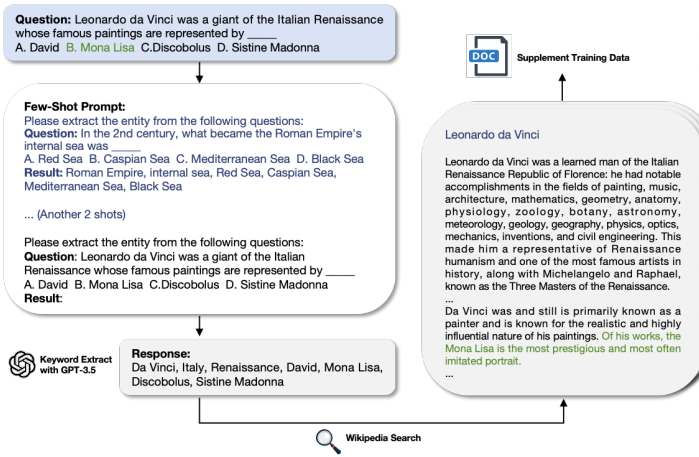
- CEM：基于LLM的错误，借助互联网生成其缺乏知识对应补充语料，使用补充训练方式完成对模型的更新，从而持续补足模型知识缺陷
- 提出了一种新颖的补充语料构建策略：
  - 加入抽取语料，使模型具有从大段语料中捕捉、提取事实知识的能力
  - 采样上一轮正确回答，保持模型对正确知识的记忆，缓解LLM对知识的遗忘

$$D_E = Construct(C_C, C_S, C_E)$$

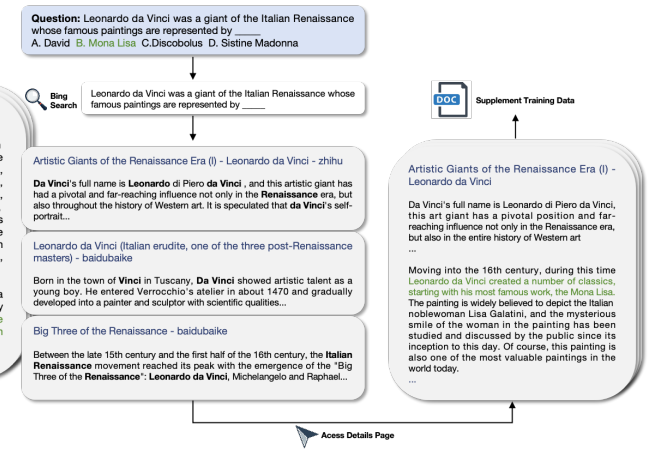
$$D_{R_i} = Construct(C_C, C_S, C_E, C_{R_i}),$$



The complete pipeline of CEM



Process of CEM based Wikipedia



Process of CEM based Bing

实验结果

- 使用三个LLM在两个QA数据集验证了CEM方法的有效性，模型回答的准确率均有提升

Metrics	CuteGPT-13B			Qwen-7B-Chat			ChatGLM-6B		
	Baseline	Wiki	Bing	Baseline	Wiki	Bing	Baseline	Wiki	Bing
Acc	42.93	45.91	46.56	41.62	48.71	47.87	40.44	51.95	53.61
W2R	-	22.20	21.91	-	22.17	20.94	-	26.01	25.86
R2W	-	18.57	18.93	-	15.08	14.70	-	14.50	12.69

XieZhi数据集上的实验结果

Metrics	CuteGPT-13B			Qwen-7B-Chat			ChatGLM-6B		
	Baseline	Wiki	Bing	Baseline	Wiki	Bing	Baseline	Wiki	Bing
Acc	26.44	32.30	35.32	24.72	39.12	40.80	23.00	35.62	35.26
W2R	-	23.84	25.14	-	25.24	26.94	-	25.96	26.32
R2W	-	17.98	16.26	-	11.04	11.02	-	13.34	14.06

CMMLU数据集上的实验结果

- 我们提出的补充语料构建策略显著降低了LLM在迭代完善的过程中把错误改成正确的概率。在最好的情况下，准确率提升了12.25%

Data Source	Metrics	Baseline	N	E	EC	R1	R2	R3	RC
Wiki	Acc	42.93	46.56	52.03	50.69	**55.18**	54.67	53.40	49.44
	W2R	-	22.20	24.12	23.42	**24.93**	24.61	24.82	22.90
	R2W	-	18.57	15.02	15.65	**12.68**	12.87	14.35	16.38
Bing	Acc	42.93	45.91	48.47	45.11	53.86	**54.53**	53.90	46.58
	W2R	-	21.91	23.15	21.37	24.25	**25.18**	24.99	22.28
	R2W	-	18.93	17.61	19.18	**13.32**	13.58	14.02	18.62
Mix	Acc	42.93	46.61	53.89	50.78	53.42	**54.43**	53.37	53.07
	W3R	-	21.99	25.12	23.67	24.04	**25.06**	25.00	24.81
	R3W	-	18.31	14.16	15.81	13.55	**13.46**	14.56	14.67

不同补充语料策略对性能提升的影响

增强大模型的自我完善能力

- 大语言模型在回答问题时，初始的生成结果可能存在不准确的情况，可以通过逐步完善的方式来优化生成答案的质量、提高模型回答的准确度。
- 方法：改变指令微调中常用的二元组<Input, Output>数据训练格式，我们提出了一种新的训练数据格式<Input, Output, Refined Output>从而赋予模型能够基于上一次的答案生成更高质量答案的能力

增强大模型的自我完善能力

- Step1:使用通用领域数据集对同一问题生成不同推理答案路径。使用不同参数大小的LLM在所选数据集上进行推理
- Step2: 根据模型参数大小，对推理答案进行排序。
- Step3:构建自我完善指令微调数据格式。新的三元组训练数据格式：

$$\langle Input,\ Output,\ \ Refined\ Output\ \rangle$$

- Step4: 进行微调训练

实验结果（初步）

✓使用llama2-13b在常识推理任务上进行了测试：

（1）base模型不具备自我完善的能力，第二次回答时准确率反而会降低

（2）我们的方法可以激发大模型自我完善的能力，第二次回答时能够提升将近5%的准确率

	Initial Answer	Second Answer
llama2-13b-chat （base）	44%	41%
llama2-13b-chat-refine（ours）	55%	**57%（+2%）**
llama2-13b-chat-ift	55%	/

检索增强的代码生成

- 基于检索与目标功能相似代码并进行改写（检索—复用—改写）是人类程序员常用的一种方式。代码大模型也可以利用这种方式来进一步提升代码生成的质量。
- 目前基于检索增强代码大模型的相关工作中，**忽略了检索代码与正确答案代码之间的关联性并未充分利用检索代码来辅助生成答案**，仅仅将其作为prompt中的一部分。
- 挑战：
  - 1. 生成代码策略：如何使用检索代码辅助生成多个代码结果
  - 2. 选取最优输出结果策略：如何在多个有差异的代码结果中选择最优的作为模型最终的生成答案

```
// Language: Java
//     void push(boolean async) {
//     if (!isConnected()) {
//         if (async && state != State.RESPONSE_PENDING) {
//             state = State.PUSH_PENDING;
//         } else {
//             state = State.RESPONSE_PENDING;
//         }
//     } else {
//         try {
//             UI ui = VaadinSession.getCurrent().getUIById(this.uiId);
//             Writer writer = new StringWriter();
//             new UidlWriter().write(ui, writer, async);
//             sendMessage("for (;;) ; [{ " + writer + " }]");
//         } catch (Exception e) {
//             throw new PushException("Push failed", e);
//         }
//     }
// }
/*base on above complete the following code*/
    public void push(boolean async) {
    if (!isConnected()) {
        if (async && state != State.RESPONSE_PENDING) {
            state = State.PUSH_PENDING;
        } else {
            state = State.RESPONSE_PENDING;
        }
    } else {
        try {
```

Ground truth

```
            Writer writer = new StringWriter();
            new UidlWriter().write(getUI(), writer, async);
            sendMessage("for(;;);[{" + writer + "}]");
        } catch (Exception e) {
            throw new RuntimeException("Push failed", e);
        }
    }
}
```

直接补全生成结果

```
            push();
        } catch (Exception e) {
            throw new RuntimeException("Failed to push", e);
        }
    }
}
```

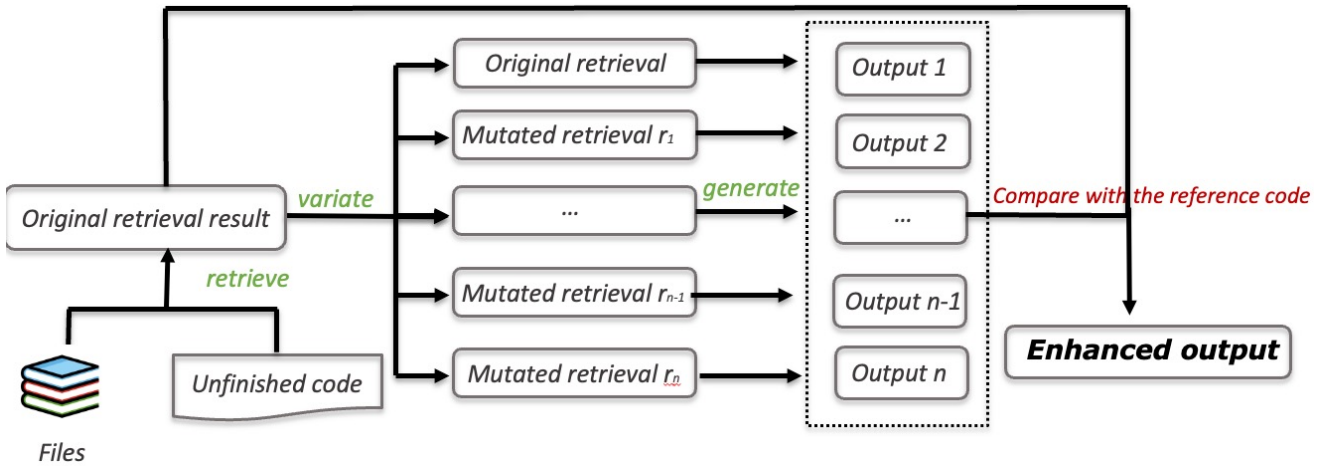
基于RAG代码补全生成结果

```
            StringWriter stringWriter=new StringWriter();
            UidlWriter uidlWriter=new UidlWriter();
            uidlWriter.write(VaadinSession.getCurrent().getUIById(this.uiId),stringWriter,asy
            sendMessage("for (;;) ; [{ "+stringWriter+" }]");
        } catch (Exception e) {
            throw new PushException("Push failed", e);
        }
    }
}
```

codegeex2-6B完成java代码补全输出时，即使不进行微调，使用RAG能够提升生成结果的质量

检索增强的代码生成

- ✓ Step1：通过retriever模块检索出与待补全代码相近的reference code。
- ✓ Step2：对reference code进行**semantic-consistent modification**,得到多个在语义上是相同的但有细微差别的reference codes。 将其与所得结果与待补全代码组合作为prompt，从而为待补全代码生成多个输出结果
- ✓ Step3：通过与reference code比较来选取最优输出结果的作为最终的生成结果



所提方法的流程图

检索增强的代码生成

- semantic-consistent modification：对代码段基于抽象语法结构树进行解析，进行例如替换参数名称，替换局部变量名称，添加无用代码段，替换运算符等不改变语义的变换。
- 选取最优输出结果策略：将待补全代码（上半段）与输出结果（下半段）拼接，并将其与检索结果比较相似度，选择相似度最高的作为最终的输出结果

```
// seed prompt
def compare(a,b):
    res = a > b
```

```
def compare(a,b):      // REL_R
    LocalVar1 = a > b
def compare(a,b):      // REL_C
    compare_res = a > b
```

semantic-consistent modification示例：替换参数名。

R后缀代表简单的replace，C后缀代表替换中包含上下文信息context（函数名）

Class	Methods	Abbreviations
Identifier Level	rename parameter regulate	REP_R
	rename parameter context	REP_C
	rename local variable regulate	REL_R
	rename local variable context	REL_C
Instruction Level	instruction replacement	IRR
	replace bool expression	RTF
Block Level	garbage code insertion regulate	GRA_R
	garbage code insertion context	GRA_C
	print statement insertion	INI

采用的Modification类型总结（共9种）

实验结果(初步)

- 使用codeGeex2-6B在AixBench-L上进行了实验
- 与其他baseline相比，我们的方法表现的最好。并且我们的输出结果选取策略是最接近理论上限值的（68%）

	Metrics	Direct generation	RAG	Internal comparison	VAR（ours）	Var-avg	ceiling
codeGeex-6B	Edit similarity	37.95	57.73	62.03	**64.55**	58.52	68
	Bleu	-	29.43	-	**36.01**	-	-
codeGeex-13B	Edit similarity	-	63.69	-	**70.11**	-	-
	Bleu	-	22.43	-	**30.28**	-	-

Thank You !