Mathematical Language Models: A Survey

韩子坚

华中师范大学计算机学院

2024 年 10 月 25 日

Content

Arithmetic Representation

- **Numerical Challenges**
  - 初期处理数值时，常被忽略或简单化。
  - BERT 在遇到数值答案时表现较差。

Arithmetic Representation

- **Numerical Challenges**
  - 初期处理数值时，常被忽略或简单化。
  - BERT 在遇到数值答案时表现较差。
- **近期表示方法**
  - GenBERT:
    - 数字按位数进行标记。
    - 进行算术问题的微调。
  - 数字转换为科学计数法。
  - 使用数字嵌入形成整体的数字表示。
  - 使用 Digit-RNN 和指数嵌入，重点突出指数。
  - 引入一致的标记化方法，增强相似数值之间的关系。

Arithmetic Calculation

- **近期方法**
  - 应用 specialized prompt engineering 提升加法能力，但乘法有位数限制。

Arithmetic Calculation

- **近期方法**
  - 应用 specialized prompt engineering 提升加法能力，但乘法有位数限制。
  - 利用 relative position embeddings 和 training set priming 实现算术任务的长度泛化。

Arithmetic Calculation

- **近期方法**
  - 应用 specialized prompt engineering 提升加法能力，但乘法有位数限制。
  - 利用 relative position embeddings 和 training set priming 实现算术任务的长度泛化。
    - （回忆一下 Transformer 的 encode 过程）一般来说，Transformer 模型使用绝对位置嵌入来为输入序列中的每个位置提供一个固定的位置标识符，这种方式在处理序列长度变化时表现不佳，因为这种嵌入将标记的位置信息与其自身的表示混合在一起，导致模型对序列长度的变化非常敏感。

Arithmetic Calculation

- **近期方法**
  - 应用 specialized prompt engineering 提升加法能力，但乘法有位数限制。
  - 利用 relative position embeddings 和 training set priming 实现算术任务的长度泛化。
    - （回忆一下 Transformer 的 encode 过程）一般来说，Transformer 模型使用绝对位置嵌入来为输入序列中的每个位置提供一个固定的位置标识符，这种方式在处理序列长度变化时表现不佳，因为这种嵌入将标记的位置信息与其自身的表示混合在一起，导致模型对序列长度的变化非常敏感。
    - 相对位置嵌入则是用来编码标记之间的相对距离，而不是它们的绝对位置。这意味着模型可以更容易地将学习到的模式应用到未见过的序列长度上，因为相对位置信息可以更好地适应变化的序列长度。

Arithmetic Calculation

- **近期方法**
  - 应用 specialized prompt engineering 提升加法能力，但乘法有位数限制。
  - 利用 relative position embeddings 和 training set priming 实现算术任务的长度泛化。
    - （回忆一下 Transformer 的 encode 过程）一般来说，Transformer 模型使用绝对位置嵌入来为输入序列中的每个位置提供一个固定的位置标识符，这种方式在处理序列长度变化时表现不佳，因为这种嵌入将标记的位置信息与其自身的表示混合在一起，导致模型对序列长度的变化非常敏感。
    - 相对位置嵌入则是用来编码标记之间的相对距离，而不是它们的绝对位置。这意味着模型可以更容易地将学习到的模式应用到未见过的序列长度上，因为相对位置信息可以更好地适应变化的序列长度。
    - 对于乘法任务，相对位置嵌入并不能使模型实现长度上的泛化（仅仅依靠相对位置信息可能不足以捕捉到乘法运算的所有细节）。为了克服这一限制，作者提出了“训练集引导”的方法，通过在训练集中添加少量长序列的例子来帮助模型泛化到更长的序列长度上。例如，在训练集中包含极少的长序列样本（如 50 个样本），就能使原本只能处理短序列的模型扩展到处理 35 位数乘以 3 位数这样的长序列计算任务。

Arithmetic Calculation

- ScratchpadGPT 通过 CoT 在 8 位加法中表现出色。

Arithmetic Calculation

- ScratchpadGPT 通过 CoT 在 8 位加法中表现出色。
- 监督学习用于微调大整数的基础运算。(将各种算术任务分为可学习和不可学习的任务，然后通过利用基本算术原则将不可学习的任务分解为一系列可学习的任务。)

Arithmetic Calculation

- ScratchpadGPT 通过 CoT 在 8 位加法中表现出色。
- 监督学习用于微调大整数的基础运算。(将各种算术任务分为可学习和不可学习的任务，然后通过利用基本算术原则将不可学习的任务分解为一系列可学习的任务。)
- MathGLM 通过在数据集中分解复杂算术表达式，逐步生成答案并学习计算规则。

Math Problem Solving

- **Math Word Problem Solving**
  - 数学文字问题 (MWPs) **通过文字描述来呈现数学概念和计算**，要求从中提取相关数学信息并应用适当的数学原理。
  - 数学文字问题解决的研究重点在于**使用高效智能的算法解决问题**。

Math Problem Solving

- **Math Word Problem Solving**
  - 数学文字问题 (MWPs) **通过文字描述来呈现数学概念和计算**，要求从中提取相关数学信息并应用适当的数学原理。
  - 数学文字问题解决的研究重点在于**使用高效智能的算法解决问题**。
  - 近期研究：
    - MathPrompter 使用 GPT-3 DaVinci 模型取得优异成绩，展现 LLMs 在复杂数学推理方面的潜力。
    - 拒绝采样微调 (RFT) 提升推理泛化能力。(通过生成多个候选样本，选择其中高质量样本进行模型微调的训练方法)
    - MetaMath 通过从多个角度重新编写数学问题来生成新的数据集 MetaMathQA，然后在此数据集上微调 LLaMA-2 模型

Math Problem Solving

- MathAttack:
  - 逻辑实体识别: 找出题目中的关键元素，比如人物（Role Entity）、数值（Number Entity）以及场景（Scenario Entity，如时间或地点）。这些元素对于构成题目逻辑至关重要，因此一旦改变就可能会改变题目的实际意义。
  - 冻结逻辑实体: 这些重要的信息将不会被更改
  - 词级攻击: 不改变上述逻辑实体的前提下，对题目中的其他词汇进行替换或修改，以期影响模型对题目的理解和解答能力。
  - 这种方法旨在创造一个对抗样例，这个样例在人类看来与原题非常相似，但对于机器学习模型来说，它可能导致错误的理解或解答。提高了 LLMs 对于数学问题的鲁棒性。

Math Problem Solving

- MathAttack:
  - 逻辑实体识别: 找出题目中的关键元素，比如人物（Role Entity）、数值（Number Entity）以及场景（Scenario Entity，如时间或地点）。这些元素对于构成题目逻辑至关重要，因此一旦改变就可能会改变题目的实际意义。
  - 冻结逻辑实体: 这些重要的信息将不会被更改
  - 词级攻击: 不改变上述逻辑实体的前提下，对题目中的其他词汇进行替换或修改，以期影响模型对题目的理解和解答能力。
  - 这种方法旨在创造一个对抗样例，这个样例在人类看来与原题非常相似，但对于机器学习模型来说，它可能导致错误的理解或解答。提高了 LLMs 对于数学问题的鲁棒性。
- LLEMMA:
  - **MATH+Python**: 模型需交替地用自然语言描述解题步骤，然后编写相应的 Python 代码来执行该步骤。最后的答案是一个数字结果或者 SymPy 对象。

Theorem Proving

Baldur:

1. **证明生成**: 将定理陈述作为输入，LLM 尝试生成完整的证明，然后使用 Isabelle 证明助手进行验证。
2. **证明修复**: 如果验证结果表明证明失败，则使用 LLM 尝试新的证明，并再次使用 Isabelle 证明助手进行验证。
3. **上下文信息**: 将定理所在的上下文信息作为补充输入，以帮助 LLM 生成更准确的证明。

Theorem Proving

Baldur:

1. **证明生成**: 将定理陈述作为输入，LLM 尝试生成完整的证明，然后使用 Isabelle 证明助手进行验证。
2. **证明修复**: 如果验证结果表明证明失败，则使用 LLM 尝试新的证明，并再次使用 Isabelle 证明助手进行验证。
3. **上下文信息**: 将定理所在的上下文信息作为补充输入，以帮助 LLM 生成更准确的证明。

Draft, Sketch, and Prove(DSP):

1. **起草非正式证明**：从一个数学问题的非正式陈述开始，这个陈述通常是以自然语言和数学公式混合的形式描述的。
2. **映射成形式证明草图**：使用大型语言模型将每个非正式证明自动转换为形式证明草图。
3. **证明剩余的猜想**：使用这些形式证明草图来引导自动化证明器，导向更简单的子问题。

Instruction Building

Instruction Building 是利用 LLMs 创建数据集的阶段。
WizardLM 使用了 Evol-Instruct 方法让 LLM 自动生成高质量的 Instructions.

Evol-Instruct 从广度、深度、增加约束条件、具象化等方面演化。

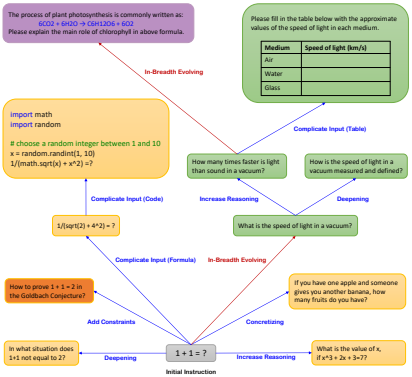


图 1: Example of Evol-Instruct taken from WizardLM

Reinforcement Learning from Evol-Instruct Feedback (RLEIF)

RLEIF 在 Evol-Instruct 的基础之上增加了 Process-supervised Reward Model (PRM) 作为监督模型，在 proximal policy optimization (PPO) 阶段会综合考虑这两个 reward model 的得分。

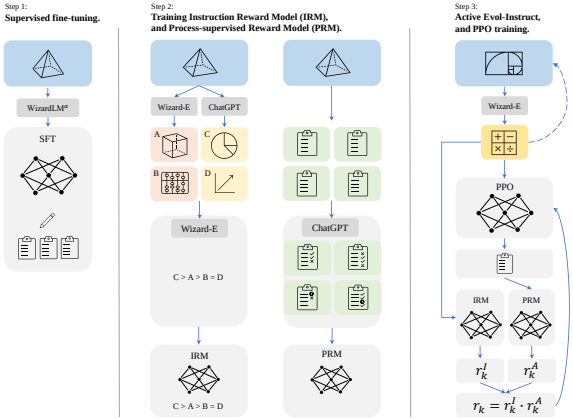


图 2: A diagram illustrating the three steps of RLEIF

Instruction Tuning

PaLM 2-L-Math:

- 发现 step-by-step 对于解题的质量提升有明显的效果。
- 综合使用 solution re-ranking 与 majority voting 会比单独使用效果更好。
- 通过多任务微调的方法，将生成解答和评估解答这两个任务分开，可以获得比只对解答进行单一微调更好的效果。

In-context Learning (ICL)

ICL 是指通过在推理时提供特定任务示例，而无需更新模型参数。

- **ScratchpadGPT:** 要求模型在 Scratchpad 中输出中间计算步骤，提升了正确率。

In-context Learning (ICL)

ICL 是指通过在推理时提供特定任务示例，而无需更新模型参数。

- **ScratchpadGPT:** 要求模型在 Scratchpad 中输出中间计算步骤，提升了正确率。
- **Codex-math:** 通过生成代码的方式结合少样本学习自动生成解决数学问题的代码。

In-context Learning (ICL)

ICL 是指通过在推理时提供特定任务示例，而无需更新模型参数。

- **ScratchpadGPT:** 要求模型在 Scratchpad 中输出中间计算步骤，提升了正确率。
- **Codex-math:** 通过生成代码的方式结合少样本学习自动生成解决数学问题的代码。
- 选择复杂且多样化的示例可以提升推理性能。

Single-tool Methods

SymbLLM 将语言模型与符号求解器结合，自动生成高质量的数学文字题逐步解答，专注于数学应用中的问题。

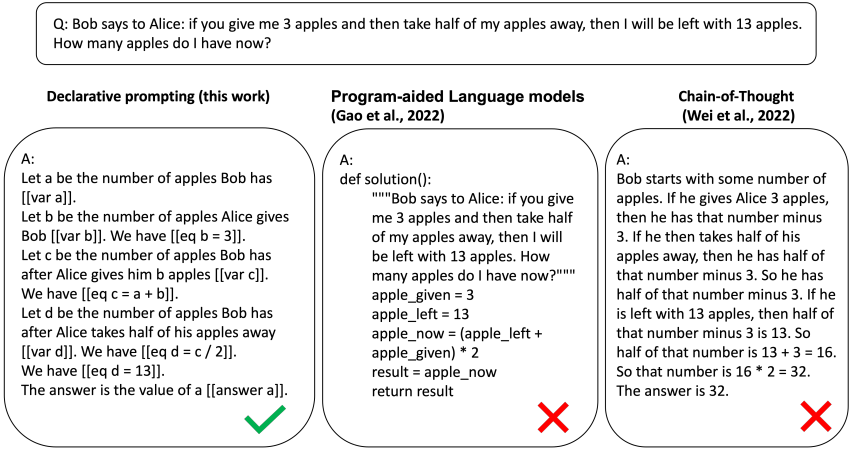


图 3: 三种方式的对比

Single-tool Methods

PoT 将 CoT 与程序相结合，提高计算的准确度。

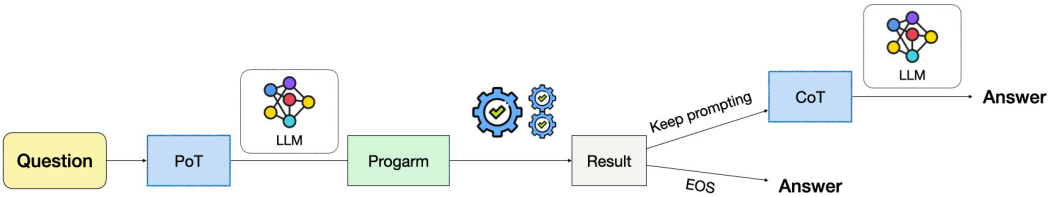
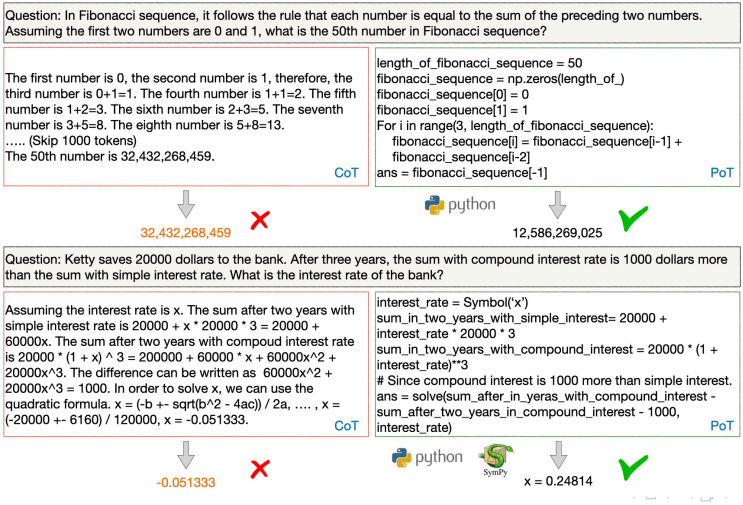


图 4: Prompting language models to first generate an intermediate answer and then continue to prompt large models to generate the final answer.

Single-tool Methods



Multi-tool Methods

Toolformer：

- Toolformer 通过少量的人工标注示例学习如何调用 API，包括决定何时调用 API、传递哪些参数以及如何将结果融入到未来的预测中。
- 这一过程是自监督的，意味着模型可以通过自身的预测来确定哪些 API 调用是有帮助的，并基于这些有用的 API 调用进行微调，使得 LLM 能够自主决定何时以及如何使用哪种工具，从而实现更加全面的工具利用，而不仅仅局限于特定的任务。

Multi-tool Methods

Toolformer:

- Toolformer 通过少量的人工标注示例学习如何调用 API，包括决定何时调用 API、传递哪些参数以及如何将结果融入到未来的预测中。
- 这一过程是自监督的，意味着模型可以通过自身的预测来确定哪些 API 调用是有帮助的，并基于这些有用的 API 调用进行微调，使得 LLM 能够自主决定何时以及如何使用哪种工具，从而实现更加全面的工具利用，而不仅仅局限于特定的任务。

Chameleon:

- **Module Inventory**: Chameleon 框架有一个模块库存，里面有多种外部工具的丰富模块库存。这些工具包括但不限于知识检索、查询生成器、行查找、列查找、表格言语化、OpenAI 程序生成器、解决方案生成器、Hugging Face 图像描述器、GitHub 文本检测器、Web 搜索引擎（如必应搜索）、Python 程序验证器、程序执行器和基于规则的答案生成器等。
- **Planner**: 借助少量的示例，自然语言规划器（Planner）可以在每一个阶段从模块库存中选择合适的模块调用。

Multi-tool Methods

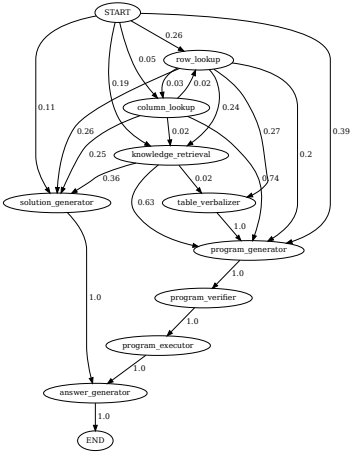


图 6: Chameleon 框架示例 1

Multi-tool Methods

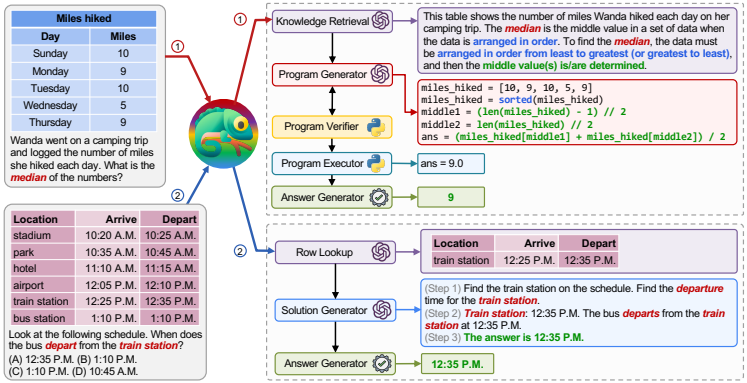


图 7: Chameleon 框架示例 2

Multi-tool Methods

ToolkenGPT 将每个工具表示为一个特殊的 token，称为 toolken 并为这个 token 学习一个嵌入向量。这样，当模型生成文本时，就像生成普通的词汇 token 一样，可以触发这些 toolken。一旦某个 toolken 被触发，LLM 就会被提示完成该工具所需的参数，从而执行具体的工具功能。这种方法可以很方便地动态添加任意数量的工具。

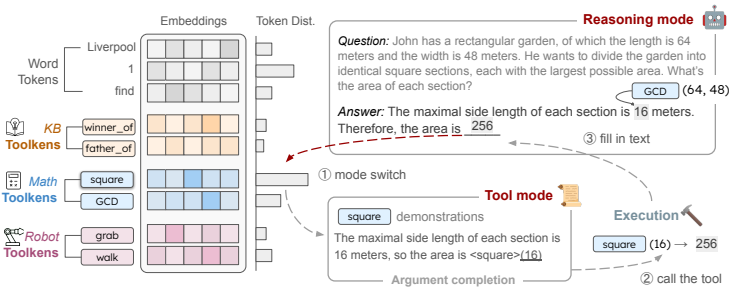


图 8: Overview of ToolkenGPT framework.

Thank you!