一、形式语言与自动机：

1、语言与问题的定义：

语言的定义：若Σ为字母表且$\forall L \subseteq \Sigma^*$，则称$L$为字母表Σ上的语言；

自然语言，程序设计语言均属于语言；

语言唯一重要的约束是所有字母表都是有穷的；

2、自动机引入：

（1）有限状态机：如 Moore Machine ， Mealy Machine；可被应用在：数字电路设计，电脑 AI 设计，各种通讯协议：TCP，HTTP，BlueTooth，wifi，文本搜索；

有穷自动机重要的是状态，最后读的是接收？还是拒绝？的状态；

DFA:确定的有穷自动机

确定的有穷自动机(*DFA, Deterministic Finite Automaton*)

A 为五元组

$$A = (Q, \Sigma, \delta, q_0, F)$$

❶ Q：有穷状态集；

❷ Σ：有穷输入符号集或字母表；

❸ $\delta : Q \times \Sigma \to Q$, 状态转移函数；

❹ $q_0 \in Q$：初始状态；

❺ $F \subseteq Q$：终结状态集或接受状态集.

$\delta(q, a) = p$

DFA 的状态转移图表示法：

❶ 每个状态 q 对应一个节点, 用圆圈表示;

❷ 状态转移 $\delta(q, a) = p$ 为一条从 q 到 p 且标记为字符 a 的有向边;

❸ 开始状态 q_0 用一个标有 start 的箭头表示;

❹ 接受状态的节点, 用双圆圈表示.

扩展的状态转移函数：

定义

扩展 δ 到字符串, 定义扩展转移函数 $\hat{\delta} : Q \times \Sigma^* \to Q$ 为

$$\hat{\delta}(q, w) = \begin{cases} q & w = \varepsilon \\ \delta(\hat{\delta}(q, x), a) & w = xa \end{cases}$$

其中 $a \in \Sigma$, $w, x \in \Sigma^*$.

那么, 当 $w = a_0 a_1 \cdots a_n$, 则有

$$\begin{aligned} \hat{\delta}(q, w) &= \delta(\hat{\delta}(q, a_0 a_1 \cdots a_{n-1}), a_n) \\ &= \delta(\delta(\hat{\delta}(q, a_0 a_1 \cdots a_{n-2}), a_{n-1}), a_n) = \cdots \\ &= \delta(\delta(\cdots \delta(\hat{\delta}(q, \varepsilon), a_0) \cdots, a_{n-1}), a_n) \end{aligned}$$

若 D 是一个 DFA，则 D 接收的语言为：

$$\mathbf{L}(D) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}.$$

如果语言L是某个 DFA 的语言，则称 L 是正则语言

NFA：非确定的有穷自动机

首先明确：什么叫状态的非确定转移？

是指同一个状态在相同的输入下可以有多个转移状态；自动机可以处在多个当前状态；使自动机的设计更容易。

若 $N=(Q,\Sigma,\delta,q_0,F)$ 是一个NFA，则 N接受的语言为

$$\mathbf{L}(N)=\{w\in\Sigma^*\mid\hat{\delta}(q_0,w)\cap F\neq\emptyset\}.$$

3、形式语言定义以及与自然语言的区别：

形式语言定义：形式语言是一种由**字母表中的符号**按照**特定的语法规则**构成的字符串集合。这些规则是精确且无歧义的，通常通过形式化的方法来定义。例如在计算机科学中，编程语言就是一种形式语言，像 python 语言，有着明确的语法规则来规定如何书写语句，定义变量等。

与自然语言的区别：

（1）在语法规则上：

形式语言的语法规则是严格且精确的，不允许有任何的模糊性。例如在正则表达式这种形式语言中，符号和组合方式都有严格规定，例如[a-zA-Z0-9]表示匹配任意一个字母或者数字，不会有其他含义。

自然语言的语法规则较为灵活，存在大量的例外，歧义现象。

（2）语义表达上：

形式语言的语义在设计语言时，通常就已经明确赋予了，是固定且单一的；

自然语言的语义依赖于语境，文化背景等多种因素，具有丰富的隐含意义与多义性。

（3）使用目的上：

形式语言主要用于计算机程序设计，数学逻辑表达，自动机理论等，强调精确性与可计算性；

自然语言是人类日常交流，表达思想情感的工具，注重信息传递的流程性与丰富性。

4、形式语法的定义与类型：

形式语法是用于精确描述形式语言结构的规则集合。它由一组**终结符号**（语言中实际出现的基本符号），一组**非终结符号**（用于表示语法结构的的抽象符号），一个**开始符号**，以及**一组产生式规则**（用于描述如何从非终结符号推导出终结符号或者其他非终结符号的组合）组成。

类型：

（1）0 型语法（短语结构文法）：产生式规则的形式为$\alpha\rightarrow\beta$，其中α和β是由终结符号与非终结符号组成的字符串，且α中至少包含一个非终结符号。0 型文法可以描述任何递归可枚举语言，表达能力最强，图灵机可以识别由 0 型文法生成的语言。

（2）1 型文法（上下文有关文法）：产生式规则的形式为$\alpha_1A\alpha_2\rightarrow\alpha_1\beta\alpha_2$，其中 A 是非终结符号，$\alpha_1$，$\beta$，$\alpha_2$是由非终结符号和终结符号组成的字符串，$\beta$不为空。意味着只有当非终结符号 A 处于上下文之间，才可以将 A 替换成β

（3）2 型文法（上下文无关文法）：产生式规则的形式为$A\rightarrow\beta$，其中 A 是非终结符号，β是由终结符号与非终结符号组成的字符串。非终结符号的替换不依赖上下文。

（4）3 型文法（正则文法）：分为右线性文法（产生式规则$A\rightarrow aB$，其中 A，B 是非终结符号，a 是终结符号）和左线性文法（产生式规则$A\rightarrow Ba$）。有限状态自动机可以识别 3 型文法生成的语言，常用于词法分析，如在编译器中识别标识符，关键字等）

5、自动机的定义，类型和应用：

自动机定义：自动机是一种数学模型，用于描述离散输入输出系统的行为。它可以根据当前状态和输入，按照预先定义的规则，转移到下一个状态，并产生相应的输出。

类型：

有限状态自动机（FSA）：由有限个状态、一个初始状态、一组终态，输入字母表以及状态转移函数构成。它只能记住有限的信息，适用于处理有限状态的问题。
下推自动机（PDA）：在有限状态自动机的基础上，增加了一个栈存储结构。它可以处理上下文无关语言，能够处理一些需要记忆更长距离信息的问题。
图灵机：是一种理论上的计算模型，具有无限的存储能力，能够模拟任何算法计算过程，是计算能力最强的自动机模型，可识别 0 型文法生成的语言。
应用：
有限状态自动机常用于词法分析，比如识别文本的单词边界，拼写检查等。**下推自动机**可用于处理一些具有嵌套结构的自然语言现象，如括号匹配，句子中的主谓宾结构嵌套等；

二、语料库与语言模型：

1、什么叫语料库：

语料库（Corpus）是按照一定的语言学原则，经过系统采集与标注的大规模自然语言文本集合。它并非简单的文本堆砌，需满足**代表性**（覆盖目标语言的不同领域）、**平衡性**（各语言变体占比合理）与**标注性**（可含词性，语法，语义等）。如 LDC 发布的语料库，是自然语言处理研究的基本资源。

2、在自然语言处理中的作用：

（1）、模型训练与优化：为语言模型提供训练数据，让模型学习语言的统计规律与语义模式。例如预训练语言模型 BERT 的训练依赖大规模语料库，通过掩码预测，句对预测，学习语言表示。
（2）、语言现象研究：助力语言学家分析词汇分布，句法结构演变，语义搭配模式等。
（3）、系统评测基准：作为测试集或验证集，评估自然语言处理系统的性能。典型如 WMT 机器翻译评测，用标准语料库衡量翻译质量。
（4）、知识挖掘与构建：从语料库中抽取词汇语义网，领域术语库等知识资源，支撑信息检索，问答系统与应用。

3、什么是语言模型？语言模型试图解决什么问题？

语言模型定义：语言模型是描述自然语言概率分布的数学模型，形式化表示为：对序列$w_1, w_2, w_3, \dots w_n$，计算联合概率$P(w_1, w_2, \dots, w_n)$，刻画该序列作为自然语言的“合理程度”，从统计语言模型到神经语言模型，核心是学习语言序列的概率规律。
试图解决问题：
（1）序列概率估计：判断自然语言序列的合法性，如检测文本是否符合语法规则与语义习惯。
（2）序列生成预测：给定前文，预测后续可能的词汇或者文本，支撑自动补全，机器翻译译词选择，对话系统回复生成等任务。
（3）语义与句法建模：通过概率分布隐含学习语言的语义关联和句法结构，为上层自然语言处理任务，提供基础表示。

4、基于 n-gram 语言模型的自动补全系统设计与问题：

预测逻辑：n-gram 模型通过统计历史 n-1 个词的条件概率预测下一个词。对于 4-gram（n=4），下一词w的概率计算为：

$$P(w|\text{"I want to eat"}) \approx \frac{Count(\text{"I want to eat"} + w)}{Count(\text{"I want to eat"})}$$

统计语料库中“I want to eat”后接各词的频次，以及这个词条本身出现的频次，按概率排序选取 top-k 候选词。
核心问题：
稀疏性：若词料库中本身词条出现的次数过少的话，可能导致分母接近于 0，导致 n-gram 模型失效；
长距离依赖：n-gram 仅仅依赖最近 n-1 个词，无法捕捉与前面较远语义的关联。
5、实现的 top-k 预测词：
首先附上执行代码后的结果：

```
Unigram 预测结果（最常见词）：[]
Bigram 预测结果（基于最后1个词 'eat' 前的 'to'）：['the', 'too', "'"]
Trigram 预测结果（基于最后2个词 'to eat' 前的 'want to'）：['his', 'up', '.']
```

原理阐述：
Unigram（1-gram）：只看当前词本身的频率，不考虑上下文，公式：

$$P(w) = \frac{Count(w)}{总词数}$$

Bigram(2-gram):看前一个词，预测下一个词，公式：

$$P(w_n|w_{n-1}) = \frac{Count(w_{n-1}, w_n)}{Count(w_{n-1})}$$

Trigram（3-gram）：看前两个词，预测下一个词，公式：

$$P(w_n|w_{n-2}, w_{n-1}) = \frac{Count(w_{n-2}, w_{n-1}, w_n)}{Count(w_{n-2}, w_{n-1})}$$

优缺点分析：
优点：简单易实现，适合小规模语料库快速构建；
缺点：稀疏性--语料库中没出现的 n-gram 会导致概率为 0，而且只能看最近 n-1 个词，无法处理复杂语义；
6、语言模型中平滑技术的必要性与风险：
（1）平滑技术的应用：
平滑是调整 n-gram 概率分布的方法，核心是为未出现的 n-gram 分配极小概率，避免因数据稀疏导致概率为 0，保证模型的鲁棒性。典型方法有 Add-1 平滑，Good- Turing 估计，Kneser-Ney 平滑等。
（2）不使用平滑的风险（以 Add-1 平滑对比为例）：
假设语料库中仅出现 “eat pizza”“eat noodles”，未出现 “eat sushi”：

- **不使用平滑**：$P(\text{sushi}|\text{eat}) = \frac{\text{Count}(\text{“eat sushi”})}{\text{Count}(\text{“eat”})} = 0$，模型会认为 “eat sushi” 是非法序列，无法正确预测。
- **使用 Add - 1 平滑**：概率调整为 $\frac{\text{Count}(\text{“eat sushi”})+1}{\text{Count}(\text{“eat”})+V}$（$V$ 是词汇表大小），为 “eat sushi” 分配极小但非零的概率，使模型保留对罕见序列的预测能力 。

本质风险是使得模型过度“自信”于训练数据，无法泛化到未预测的语言现象，在实际应用（如自动补全，机器翻译）中产生严重错误，如拒绝合理词汇，产生语法错误的文本。

三、自然语言嵌入
1、什么是词向量，为什么要用向量表示词？

词向量：是将自然语言中的**词汇**映射到**低纬实数向量空间**的一种表示方法。在这个向量空间中，每个词汇对应一个向量，向量的每个维度捕捉了词汇在语义，语法等层面的某些特征，使得语义相近的词汇在向量空间中距离相近。从数学本质来看，这是一种**分布式表示**，通过让词汇的语义信息分布在向量的各个维度上，来刻画词汇丰富的语义，以及与其余词汇的关联。

用向量表示词的**原因**：

（1）从语义建模需求看：自然语言处理任务（如文本分类，机器翻译，问答系统）需要让计算机理解**词汇的语义**，而离散的词汇符号本身难以直接体现语义关联，向量表示可以将语义信息量化；

（2）从计算可行性上看：机器学习和深度学习模型的输入，往往需要数值型数据，词向量将模型转化成数值向量后，能更方便作为**模型的输入进行训练与推理**。

（3）缓解数据的稀疏性：在自然语言中，词汇数量庞大，若用简单的离散表示，每个词汇相互独立，对于罕见词等，难以利**用上下文信息补充语义**。词向量通过利用大规模语料中词汇的共现等统计信息，语义相似的词汇**共享向量空间的特征**，一定程度上缓解了数据稀疏问题。

2、one-hot 编码：

定义：one-hot 编码是一种简单的词汇离散表示方法。对于一个包含V个不同词汇的词汇表，每个词汇可以表示成一个长为V的向量，对应词汇所在位置的元素为 1，其余位置的元素为 0；

导致维度灾难的原因：

（1）维度较大：one-hot 编码的向量维度为词汇表的大小V。在实际的自然语言处理上，词汇表的规模往往较大，这样一来，每个词汇所对向量的维度极大；

（2）语义信息缺失：one-hot 编码的向量彼此**正交**，无法体现词汇间的**语义相关性**，很多自然语言处理任务中，需要利用词汇的语义关联来建模，而 one-hot 编码无法提供这样的信息。高维度向量进行诸如相似度计算，作为模型输入等操作时，计算量急剧增加，且容易出现**过拟合**。

3、Word2Vec 提供的两种训练架构与工作原理：

Word2Vec 的两种训练架构：连续词袋模型（CBOW）与跳字模型（Skip-gram）

连续词袋模型（CBOW）：

目标与输入输出：CBOW 的目标是根据上下文词汇来预测中心词汇。输入是某个中心词一定**窗口**内的上下文词汇的词向量，输出是中心词汇的词向量。从概率模型角度，目标是最大化$P(w_t|w_{t-n},w_{t-n+1},\dots,w_{t-1},w_{t+1},w_{t+2},\dots,w_{t+n})$

流程：首先对输入的上下文词汇的 one-hot 向量（初始时采用，后续会用训练好的词向量迭代更新）进行**嵌入矩阵**查找，得到对应的**词向量**，然后将这些上下文词向量进行平均，得到一个**上下文语义表示向量**。接着通过一个**线性变换**和一个 **softmax 激活函数**，得到对中心词的**概率分布预测**。最后通过对比真实中心词汇的词向量，利用损失函数，计算预测误差，并通过反向传播算法更新嵌入矩阵和输出权重矩阵中的参数，从而使得预测更为精确。

跳字模型（Skip-gram）：

目标与输入输出：与 CBOW 相反，Skip-gram 是根据中心词汇来预测上下文词汇。输入是中心词汇的词向量，输出是其周围一定窗口内上下文词汇的词向量。对应的概率模型是最大化$P(w_{t-n},w_{t-n+1},\dots,w_{t-1},w_{t+1},\dots,w_{t+n}|w_t)$

流程：首先对输入的中心词汇的 one-hot 向量进行**嵌入矩阵**查找，得到对应的**词向量**。然后将该中心词汇通过**线性变换和 softmax 激活函数**，为每个可能的上下文词汇

计算**预测概率**，随后同样将真实的上下文词汇的 one-hot 向量作为标签，通过损失函数**计算误差并且反向传播**，来更新嵌入矩阵等参数。训练完成后，嵌入矩阵中的向量即为**词向量**。

四、大模型预训练与微调：

1、LLM 训练阶段与区别：

大语言模型（LLM）的训练通常分为预训练与微调两个阶段：

阶段区别：

预训练阶段：旨在让模型学习**通用的语言知识与表示能力**。模型在大规模的无标注文本语料上，通过**自监督学习任务**，学习词汇，语法，语义，以及世界知识等。此阶段模型不针对特定任务，是为了获取广泛的语言理解与生成基础。

微调阶段：是在预训练模型的基础上，使用**有标注的特定任务数据**，对模型进行调整，使模型能够适应**特定的下游任务**，如文本分类，问答，机器翻译等。微调阶段更侧重于让模型在**特定任务**上表现出良好的性能。

所用数据差异：

预训练数据：通常是大规模的，多样化的无标注文本数据，涵盖书籍，网页，文章等各种类型的文本，数据量可达数十亿甚至数千亿 tokens，目的是让模型尽可能学习到全面的语言规律。

微调数据：是与特定下游任务相关的有标注数据，数据量相对预训练要小很多，通常几千到几十万数量不等，数据内容围绕特定数据。

2、微调方法以及适应场景：

（1）全参数微调：

方法概述：在微调过程中，调整预训练模型的所有参数；

适用场景：当有充足的特定任务标注数据，且希望模型在该任务上达到最佳性能时适用。例如在大规模的机器翻译任务中，若有大量的高质量的平行语料，全参数微调能让模型更适配任务。

（2）参数高效微调：

LoRA：通过在预训练模型的权重矩阵上添加**低秩矩阵的乘积**，只训练这些低秩矩阵的参数，从而大大减少需要训练的参数量。

适用场景：适用于资源有限，但又需要对模型进行有效微调的场景，也可用于多任务微调，能快速适配不同任务且开销较小；

Prefix Tuning：仅优化输入序列的**前缀部分的参数**，模型的其余部分保持不变。在生成任务中，通过调整前缀来引导模型生成符合特定任务要求的内容。

适用场景：主要用于**生成类任务**，如文本生成，对话生成这类的。

Prompt Tuning：设计特定的提示文本，将**下游任务**转化成类似于预训练任务的形式，通过调整**提示**的参数，让预训练模型适应下游任务。

五、自回归 AI 和生成式 AI：

1、自回归模型及其生成概率公式：

（1）自回归模型定义：自回归模型是一种生成模型，在生成序列时，假设当前位置元素的生成概率仅依赖于之前已经生成的元素。也就是说，要生成一个长度为 n 的序列 $x_1, x_2, \ldots, x_n$，自回归模型会按照顺序，逐个生成每个元素 xi，且 xi 的生成基于 x1，x2，… , xi-1。

（2）生成概率公式：对于序列$x_1, x_2, \ldots, x_n$，其生成概率可表示为：

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, x_3, \ldots, x_{i-1})$$

2、一步一词，生成任务逐词展开的原因：

（1）为什么每一步只能生成一个词：

在自回归生成中，模型需要基于已生成的上下文来预测下一个词的概率分布。如果同时生成多个词，模型需要处理的**概率空间会急剧增大**，计算复杂度会呈指数型增长。且语言的**语义与语法结构是逐步构建**的，逐个生成词更符合人类语言生成的认知过程。

（2）生成任务逐词展开的原因：

自然语言本身是具有顺序性的序列，文字的排列顺序承载着语义与语法信息。逐词展开的生成方式能够遵循**语言的序列性**，保证文本的语义和语法正确。

且在模型的预训练阶段，模型就是以逐词预测下一个词的方式进行训练的。在推理阶段采用这个方式，能与**训练阶段的目标和方式保持一致**，从而保证模型生成的效果。

六、Chain of Thought（CoT）

1、CoT 与 Test-Time-Scaling 相关概念：

CoT：是一种提示工程技术，旨在增强大语言模型（LLM）的推理能力。其核心是引导模型进行多步骤的逻辑推理，从而提升模型在数学题求解，常识推理与符号逻辑等多步逻辑任务的表现。

Test-Time- Scaling：指在不重新训练 LLM 的前提上，借助外部机制来提高模型在测试阶段的性能，检索增强生成（RAG）是典型代表。

2、CoT 的相关对比：

zero-shot 场景下，LLM 直接输出答案，往往缺乏清晰的推理过程呈现，对于复杂问题，答案的准确性和可解释性较差；

few-shot CoT 场景下，LLM 会在输出中，展示出逐步推理的链条，先分析问题条件，再运用相关知识进行推导，最后得出结论，推理过程透明化，答案往往更加合理。

3、RAG 相关对比：

不使用 RAG 时，LLM 仅依赖自身预训练的知识进行输出。当遇到需要最新知识，专业领域知识时，输出可能存在知识过时，不准确等情况；

使用 RAG 时，LLM 能够先从外部检索源获取与问题相关的信息，再基于检索到的信息生成输出，输出内容贴合实际需求。

我的问题如下：
计算积分$\int_{-\pi}^{\pi} x * sin^5 x * arctane^x dx$
一开始直接让 AI 给出计算结果为：

\(\int_{-\pi}^{\pi} x \cdot \sin^5 x \cdot \arctan(e^x) \, dx\)再看下这个，直接给出最终答案

0

那我在给出类似例题（先不开启联网搜索）后，DeepSeek 给我的回答是：（截取部分）

这说明需要换一种思路：**分析 $f(x)+f(-x)$ 的奇偶性。**

计算 $f(x)+f(-x)$：$f(x)+f(-x)=x\cdot\sin^5 x\cdot\arctan(e^x)+(-x)\cdot\sin^5(-x)\cdot\arctan(e^{-x})$

化简 $\sin^5(-x)=-\sin^5 x$，故：

$f(x)+f(-x)=x\cdot\sin^5 x\cdot\arctan(e^x)+(-x)\cdot(-\sin^5 x)\cdot\arctan(e^{-x})=x\cdot\sin^5 x\cdot[\arctan(e^x)+\arctan(e^{-x})]=x\cdot\sin^5 x\cdot\frac{\pi}{2}$

而 $f(x)-f(-x)$ 是奇函数（因为 $f(x)$ 整体既非奇也非偶），但在对称区间上，奇函数的积分为 0，即 $\int_{-\pi}^{\pi}[f(x)-f(-x)]dx=0$，故 $\int_{-\pi}^{\pi}f(x)dx=\int_{-\pi}^{\pi}f(-x)dx$，这与步骤 2 一致。

但关键结论：**$x\cdot\sin^5 x$ 是偶函数，而 $x\cdot\sin^5 x\cdot\frac{\pi}{2}$ 的积分是否为 0?**

不，显然不为 0。这说明之前的推导正确，但需要计算 $\int_{-\pi}^{\pi}x\cdot\sin^5 x dx$：

利用对称区间和函数奇偶性的更深层性质：

- 令 $t=-x$，则

  $\int_{-\pi}^{\pi}x\cdot\sin^5 x dx=\int_{-\pi}^{\pi}(-t)\cdot\sin^5(-t)(-d$ $\int_{-\pi}^{\pi}(-t)\cdot(-\sin^5 t)(-dt)=-\int_{-\pi}^{\pi}t\cdot\sin^5 t dt$

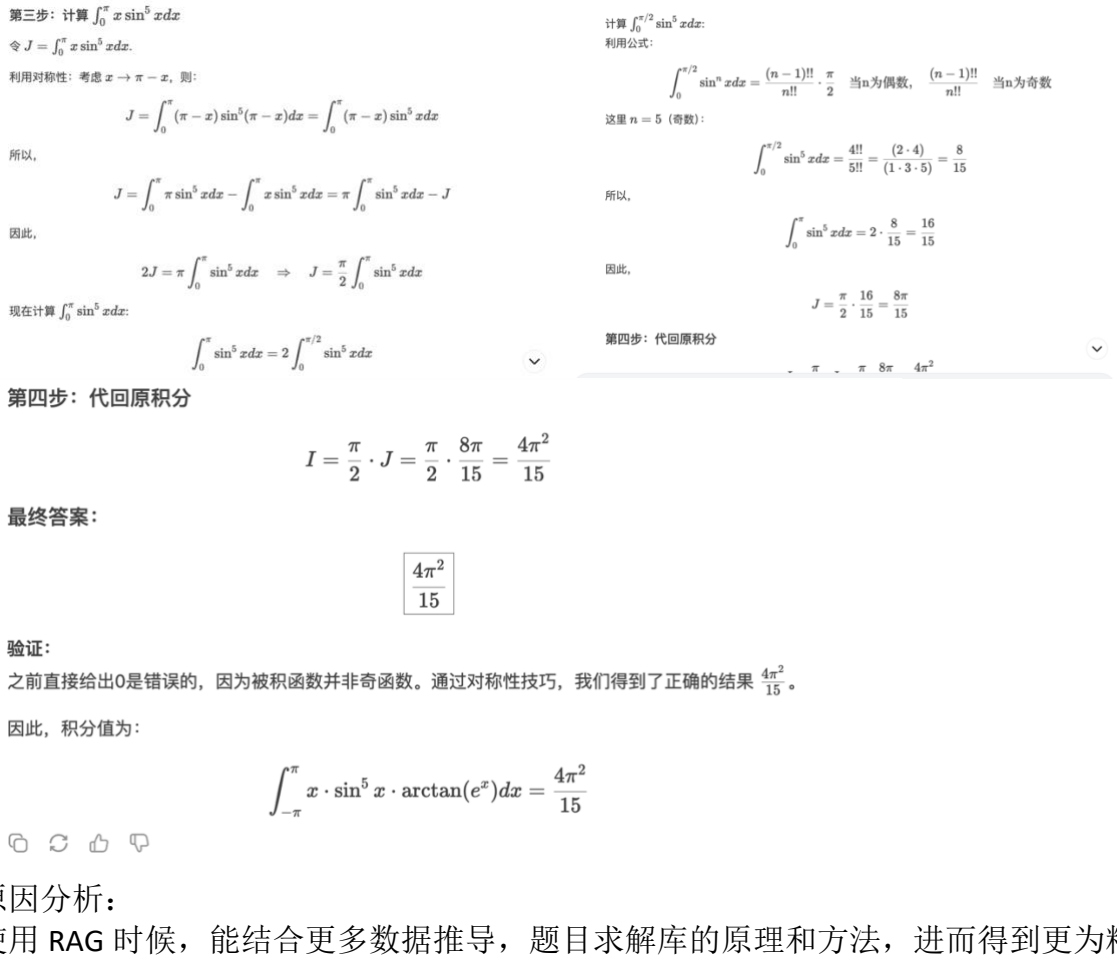
  即 $\int_{-\pi}^{\pi}x\cdot\sin^5 x dx=-\int_{-\pi}^{\pi}x\cdot\sin^5 x dx$，因此：

步骤 5：得出最终结果

将上述结论代入式 (2)：$2I=\frac{\pi}{2}\times 0=0\implies I=0$

答案： $\boxed{0}$

很遗憾虽然最终结果仍然有误，但起码这次有在严格逻辑推理，这说明了：
few-shot-CoT 通过示例提供了推理的模版，引导模型按步骤分析求解决；
再开启联网搜索后，Deepseek 这次给出了正确答案：



第三步：计算 $\int_0^\pi x\sin^5 xdx$

令 $J=\int_0^\pi x\sin^5 xdx$.

利用对称性：考虑 $x\to\pi-x$，则：

$$J=\int_0^\pi(\pi-x)\sin^5(\pi-x)dx=\int_0^\pi(\pi-x)\sin^5 xdx$$

所以，

$$J=\int_0^\pi\pi\sin^5 xdx-\int_0^\pi x\sin^5 xdx=\pi\int_0^\pi\sin^5 xdx-J$$

因此，

$$2J=\pi\int_0^\pi\sin^5 xdx\quad\Rightarrow\quad J=\frac{\pi}{2}\int_0^\pi\sin^5 xdx$$

现在计算 $\int_0^\pi\sin^5 xdx$:

$$\int_0^\pi\sin^5 xdx=2\int_0^{\pi/2}\sin^5 xdx$$

计算 $\int_0^{\pi/2}\sin^5 xdx$:

利用公式：

$$\int_0^{\pi/2}\sin^n xdx=\frac{(n-1)!!}{n!!}\cdot\frac{\pi}{2}\quad\text{当n为偶数},\quad\frac{(n-1)!!}{n!!}\quad\text{当n为奇数}$$

这里 $n=5$（奇数）：

$$\int_0^{\pi/2}\sin^5 xdx=\frac{4!!}{5!!}=\frac{(2\cdot4)}{(1\cdot3\cdot5)}=\frac{8}{15}$$

所以，

$$\int_0^\pi\sin^5 xdx=2\cdot\frac{8}{15}=\frac{16}{15}$$

因此，

$$J=\frac{\pi}{2}\cdot\frac{16}{15}=\frac{8\pi}{15}$$

第四步：代回原积分

第四步：代回原积分

$$I=\frac{\pi}{2}\cdot J=\frac{\pi}{2}\cdot\frac{8\pi}{15}=\frac{4\pi^2}{15}$$

最终答案：

$$\boxed{\frac{4\pi^2}{15}}$$

验证：

之前直接给出0是错误的，因为被积函数并非奇函数。通过对称性技巧，我们得到了正确的结果 $\frac{4\pi^2}{15}$。

因此，积分值为：

$$\int_{-\pi}^{\pi}x\cdot\sin^5 x\cdot\arctan(e^x)dx=\frac{4\pi^2}{15}$$

原因分析：

使用 RAG 时候，能结合更多数据推导，题目求解库的原理和方法，进而得到更为精确的结果。

七、LLM Mathematical Reasoning With Lean4

1、Lean4 是什么？为什么要用 Lean4？

（1）首先解释什么叫函数式编程泛式的现代交互定理证明器：

函数式编程泛式：以函数为核心，强调数据不可变，**计算过程通过函数组合**实现，避免状态修改，逻辑严谨性与数学推理天然契合，为定理形式化表达提供简洁、可靠的语法基础。

交互式定理证明器：结合**人工引导与机器验证**的工具，用户逐步构建证明步骤，系统实时检查每一步的逻辑正确性，支持复杂数学命题的形式化验证。

现在可给出 Lean4 的定义：

Lean4 是一款基于**函数式编程泛式的现代交互式定理证明器**，是 Lean 系列的最新版本，核心功能是将数学定理与证明转化为计算机可解析的形式化语言，并通过内置逻辑内核**自动验证**证明的正确性。其语法体系严格且支持**模块化开发**，同时兼容数学领域的通用定义，可依托开源库（Mathlib4）实现复杂数学命题的形式化表达。

（2）为什么用 Lean4:

Lean4 近年来获得数学界的广泛关注，其配套库（Mathlib4）包含 10 万多条人工编写的高质量数学定理与证明，覆盖基础数学至高阶领域，为形式化证明研究提供了稀缺的高质量数据基础。

且研究缺口显著：相较于 Lean3 与 Isabelle，Lean4 语法更简洁，但与自然语言差异显著，导致现有 LLM 因预训练数据中 Lean3 干扰，以及 **NL-FL 对齐数据缺乏**，在 Lean4 形式化对齐任务上性能不佳，存在明确的**技术突破空间**。
再者，Lean4 的逻辑内核支持高效的证明验证，且其形式化表达更贴近现代数学研究范式，有助于桥接“人类数学推理”与“计算机可验证证明”。

2、本工作数据集的生成流程与内容构成：

本工作提出 OBT 数据集，通过“提取-非形式化-自举”三步流程生成，具体如下：

（1）Lean4 证明提取：

从 LeanDojo 仓库中提取 Mathlib 的 10 万条 Lean4 定理与证明，涵盖逻辑，代数，拓扑等领域，确保数据的数学严谨性与覆盖度—这些数据是后续 NL-FL 对齐的基础，但本身不含 NL 描述。

（2）带示例检索的非形式化：

为解决 Mathlib4 无 NL 描述的问题，首先需要微调 ByT5 编码器；

ByT5 编码器是基于 ByT5 模型（Byte-level Transformer 5）的编码组件，主要用于文本的表示学习与语义匹配。

其核心特点在于以字节（byte）为输入单位，而非传统的子词（subword），能直接处理任意字符集，避免了特定语言子词分词带来的偏差，增强了跨语言和低资源语言处理的适应性。

以 NL 标注的 MiniF2F 数据集为训练数据，通过对比损失

$$\mathcal{L} = 1 - \cos(x_{NL}, x_{FL}) + \frac{1}{2}[\cos(x_{NL}^{-}, x_{FL}) + \cos(x_{NL}, x_{FL}^{-})])$$

对齐 NL 与 Lean4 语言的余弦相似度，实现“按相似度检索**高质量示例**”；
随后，以检索到的示例为上下文，调用 Gemini-1.5 生成 Mathlib4 中每条 Lean4 证明对应的 NL 定理陈述与证明，并通过质量检查，最终得到**初始 NL-FL 对齐数据**。

Gemini-1.5 是谷歌（Google）旗下 DeepMind 团队开发的大型语言模型，属于 Gemini 系列的重要迭代版本。

其核心特点包括更强的上下文处理能力，能够处理更长的文本序列，这使其在长文档理解、多模态信息整合（如文本、图像、音频等）方面表现突出。在自然语言处理任务中，它在复杂推理、知识问答、内容生成等场景下展现出较高的性能，尤其在需要结合多领域知识进行深度分析的任务中，优势较为明显。

在学术研究领域，如前文提及的将形式语言证明转化为自然语言证明等场景中，Gemini-1.5 因生成内容更贴合专业表述习惯而被选用，体现了其在专业文本处理上的适配性。

（3）NL-FL 自举：

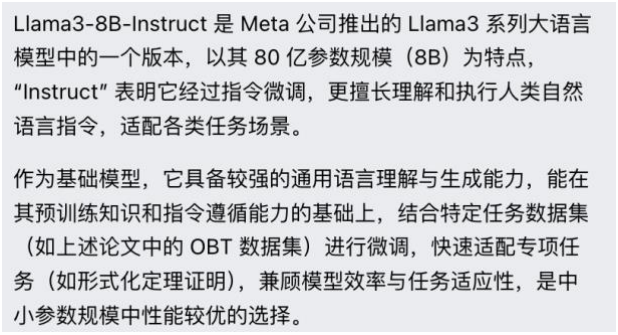
为建立 NL 推理与形式化推理的关联，将生成的 NL 证明以**代码注释形式**嵌入对应的 Lean4 代码中，并通过算法验证：移除注释后的 Lean4 代码与原始代码完全一致，确保形式化正确性。此步骤实现了“NL 推理指导 FL 代码的**自举效果**”，是数据集的核心创新。

（4）数据集内容构成：
OBT 数据集共包含 106852 条 NL-FL 对齐且经过自举处理的记录，每条记录结构如下：
基础信息：定理名称，Lean4 定理陈述，Lean4 完整证明，文件路径，Git Commit 号；
生成信息：Gemini-1.5 生成的 NL 定理陈述与 NL 证明；
自举信息：带 NL 证明注释的 Lean4 代码

3、自然语言推理与 Lean4 形式化推理的壁垒克服：

本工作通过“数据层-训练层-推理层”三级协同策略，突破 NL 与 Lean4 的推理壁垒，核心创新在于 NL-FL 自举机制，具体如下：
（1）数据层：NL-FL 自举—建立语义关联：
将 NL 证明以注释形式嵌入 Lean4 代码，使 LLM 在训练中同时接触“NL 推理步骤”与“对应的 FL 代码实现”。此设计让 LLM 学习“NL 推理意图-FL 语法映射”，解决了“NL 与 Lean4 因语法差异导致的推理迁移失效问题”，在论文 3.4 展示移除了自举后，MiniF2F-test 的准确率由 33.61%降低至 26.23%；
（2）数据层：示例检索—降低 NL-FL 映射难度：
通过微调 ByT5 编码器发现“NL 语句与 Lean4 语句的相似度匹配”，为 Gemini-1.5 生成 NL 描述时，通过**高质量上下文示例**，避免 LLM 因预训练中 Lean3 数据干扰而生成错误的 FL 映射。该编码器可有效区分相似与不相似的 NL-FL 对。
（3）训练层：块训练—增强上下文推理能力：
将训练数据视为“文本环”，在输入中整合前 k 个 NL-FL 对齐样本，例如第 i 个样本的输入为“$NL_{i-k}NF_{i-k}\dots NL_i$”，输出为$FL_i$。此设计让 LLM 在训练中学习“多步 NL 推理-多步 FL 实现”的上下文关联，提升对复杂证明的逻辑连贯性理解。
（4）推理层：迭代式证明编写—缩小分布差距：
初始以 Yang et al. (2024) 的已验证证明为上下文示例，让 LLM 生成 Lean4 证明，随后将 Lean4 验证正确的证明作为下一轮的示例，迭代至无新增证明，或者达到最大步数。论文 3.5 显示迭代 2 轮后，MiniF2F 的正确率由 31.15%提升至 33.61%；

4、本工作的模型训练过程：


Llama3-8B-Instruct 是 Meta 公司推出的 Llama3 系列大语言模型中的一个版本，以其 80 亿参数规模（8B）为特点，“Instruct”表明它经过指令微调，更擅长理解和执行人类自然语言指令，适配各类任务场景。

作为基础模型，它具备较强的通用语言理解与生成能力，能在其预训练知识和指令遵循能力的基础上，结合特定任务数据集（如上述论文中的 OBT 数据集）进行微调，快速适配专项任务（如形式化定理证明），兼顾模型效率与任务适应性，是中小参数规模中性能较优的选择。

本工作以 Llama3-8B-Instruct 为基础模型，基于 OBT 数据集进行“指令微调+迭代优化”，训练过程分为“核心训练阶段”与“训练增强阶段”，具体如下：
（1）核心训练阶段：基于 OBT 的指令微调：
数据预处理：采用**课程数据排序**，以证明所需步骤数衡量难度，将 OBT 数据按“易到难”排序，避免 LLM 因初始接触复杂证明而训练不稳定。
训练策略：
输入设计：NL 定理陈述+NL 证明+Lean4 定理陈述
输出设计：**NL-FL 自举后的 Lean4 完整证明**
训练参数：自回归训练，1000 步热身，学习率 1e-5，8 张 A6000GPU 训练约 32 小时

核心技术：融合块训练与课程排序，确保 LLM 逐步掌握 Lean4 语法与 NL 推理的映射。
（2）性能增强阶段：**迭代式证明**编写：
训练后，通过两轮迭代优化性能：
第 1 轮，以 Yang et al. (2024) 的已验证明为上下文示例（由 LLM 上下文长度决定），生成 MiniF2F 的数据集的 Lean4 证明，用 Lean 内核验证其正确性。
第 2 轮，将第一轮中正确的证明作为新示例，重新生成其余未证明定理的证明，进一步提升准确率。
5、除推理任务外 LLM 的其他应用场景：
本工作中 LLM 不仅用于最终的 Lean4 证明生成，还在**数据集构建**，**示例检索**，**基线验证**等关键环节发挥核心作用。
（1）数据集生成：NL 描述生成与异常值处理：
NL 非形式化：调用 Gemini-1.5 将 Lean4 证明转化为 NL 定理陈述与证明，Gemini-1.5 生成的 NL 证明更符合数学界表述习惯。
异常数据处理：对 Gemini-1.5 生成的异常数据，通过**迭代查询** Gemini-1.5 重新生成，确保 OBT 数据集的质量。
（2）示例检索：编码器微调与相似度匹配：
微调 **ByT5-Tacgen** 模型作为示例检索编码器：以 NL-FL 对比损失训练，使其能计算 NL 语句与 Lean4 语句的余弦相似度，为非形式化过程筛选高质量上下文示例；
（3）基线验证：性能对比与方法有效性验证

基线模型选择：采用**GPT-4-Turbo**、**Gemini-1.5-pro**、**Llemma-31B**、**DeepSeek-Math-7B**等 LLM 作为基线，验证 TheoremLlama 的优越性 —— 例如，TheoremLlama 在 MiniF2F-Test 的准确率（33.61%）显著高于 GPT-4（22.95%）与 DeepSeek-Math-7B（24.60%）（表 1）；

框架通用性验证：将 TheoremLlama 框架应用于**DeepSeek-Math-7B**（附录 G），微调后其 MiniF2F-Test 准确率从 24.60% 提升至 35.66%，证明框架对不同 LLM 的适配性，而 DeepSeek-Math-7B 本身是预训练数学 LLM，此处用于验证方法的通用性。