看视频，论文阅读

夏同

2026 年 1 月 29 日

目录

第一章　B 站 LLM 学习笔记

1.1　人工智能的演进与范式转移

人工智能（Artificial Intelligence, AI）的终极目标是创造能够执行通常需要人类智能的任务的机器或软件。其发展历程经历了

- 基于规则的“符号主义”（系统完全依赖于专家手工编写的 if-then（如果-那么）规则库，给它一本厚厚的“操作手册”或“推理规则词典”）
- 依赖特征工程的“机器学习”（核心是“从数据中学习”，而非“被规则规定”，我们不再直接编写规则，而是为机器提供描述数据的“特征”，让机器自己从这些特征和数据标签的关系中学习出模型规则）
- 能够自动学习特征表示的“深度学习”（模型不再依赖手工设计的特征，而是自动从原始数据中学习出有意义的、层次化的特征表示），最终演变为当前以“大模型“为核心的“预训练 + 生成”新范式。

这一演进的核心驱动力是“规模效应”（Scaling Laws）。大量研究表明，随着模型参数数量、训练数据量和计算量的同步、大规模增长，模型的性能会以可预测的方式显著提升，并涌现出小规模模型所不具备的复杂能力，如推理、代码生成和跨模态理解。本笔记旨在系统性地剖析这一以“大模型”为代表的 AI 新范式。

1.2　人工智能的核心子领域与层次结构

人工智能是一个伞状概念，其技术实现呈现出清晰的层次化结构：

人工智能技术体系呈现金字塔式的层次结构，从上到下依次为：

- **人工智能 (AI)**：最顶层，涵盖所有模拟人类智能的技术领域。
- **机器学习 (Machine Learning)**：AI 的核心实现方式，使计算机能够从数据中学习而不需要显式编程。
- **深度学习 (Deep Learning)**：机器学习的一个子集，使用多层神经网络自动学习数据的层次化表示。
- **生成式人工智能 (Generative AI)**：深度学习的一个快速增长分支，专注于创造新的、与训练数据相似的内容。
- **基础模型/大模型 (Foundation Models)**：在超大规模数据上预训练、可适应广泛任务

的深度学习模型，是生成式 AI 当前的主要技术实现。

每一层都建立在下一层的基础之上，体现了技术发展的递进关系。

1.2.1 机器学习：从数据中学习的科学

机器学习是 AI 的核心实现方式，其核心思想是：计算机程序通过经验（数据）自动改进其在某项任务上的性能。根据学习范式，主要分为三类：

监督学习 (Supervised Learning)

- **定义：**算法从已标注（即有明确输入-输出对）的训练数据中学习一个映射函数 $f : X \to Y$。目标是使 f 能够对新的输入 x_{new} 预测出尽可能接近真实值 y_{new} 的输出。
- **数学表述：**给定训练数据集 $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$，学习一个模型 $f(x; \theta)$，通过最小化损失函数 $L(\theta) = \sum_{i=1}^{N} \ell(f(x_i; \theta), y_i)$ 来优化参数 θ。
- **典型任务：**分类（如图像识别）、回归（如房价预测）。
- **类比：**老师提供标准答案（标签）的习题册，学生通过练习学会解题。

无监督学习 (Unsupervised Learning)

- **定义：**算法从未标注的数据中寻找内在的结构、模式或分布。没有预设的“正确答案”。
- **典型任务：**
  - **聚类 (Clustering)：**将数据点分组，使得组内相似度高，组间相似度低。
  - **降维 (Dimensionality Reduction)：**将高维数据映射到低维空间，保留主要信息（如 PCA，t-SNE）。
  - **关联规则学习：**发现数据集中变量之间的有趣关系。
- **类比：**给学生一堆未分类的动植物图片，让他们自己找出分类规律。

强化学习 (Reinforcement Learning, RL)

- **定义：**智能体（Agent）通过与环境互动来学习最优策略。其核心是探索与利用（Exploration vs. Exploitation）的权衡。类似于让机器学会条件反射。
- **核心要素：**
  - **状态 (State)** s: 对环境的描述。
  - **动作 (Action)** a: 智能体可以执行的操作。
  - **奖励 (Reward)** r: 环境对智能体动作的反馈。
  - **策略 (Policy)** $\pi(a|s)$: 智能体在给定状态下选择动作的规则。
  - **价值函数 (Value Function)** $V^{\pi}(s)$: 在状态 s 下，遵循策略 π 所能获得的长期累积奖励的期望。
- **目标：**学习一个最优策略 π^*，以最大化期望累积奖励 $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$，其中 $\gamma \in [0, 1]$ 是折扣因子。

- **类比：**训练宠物。宠物（智能体）做出动作（坐下、握手），主人根据动作给予奖励（零食）或惩罚（没有零食）。宠物通过反复尝试学习如何获得最多零食。

1.2.2　深度学习：神经网络的深度进化

深度学习是机器学习的一个分支，其核心是使用包含多个隐藏层的深度神经网络（DNNs）来学习数据的层次化表示。

人工神经元与多层感知机

最基本的单元是人工神经元（或称感知机），它模拟生物神经元：

$$y = \phi(\sum_{i=1}^{n} w_i x_i + b) = \phi(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{1.1}$$

其中 $\boldsymbol{x}$ 是输入向量，$\boldsymbol{w}$ 是权重向量，b 是偏置，ϕ 是非线性激活函数（如 ReLU, Sigmoid）。

将多个神经元按层连接，就构成了前馈神经网络（Feedforward Neural Network）或多层感知机（MLP）。深度则意味着有很多这样的隐藏层。

关键架构

- **卷积神经网络 (Convolutional Neural Network, CNN)：**
  - **主要用途：**专门处理像图片这样的网格状数据（每个像素点就是一个数据）。
  - **核心操作：**
    - **卷积：**想象你拿着一把小刷子（卷积核），在整张图片上一点点滑动。每停一个地方，就计算一下这个小区域的特征（比如有没有边缘、有没有特定纹理）。这把“小刷子”就是来提取局部特征的。
    - **池化：**可以理解为“看缩略图”或“总结摘要”。比如把一个 2x2 的小区域，只保留最重要的一个数值（比如最大值），这样既能保留关键信息，又能让数据量变小，变得更抗干扰（比如图片稍微平移一点，识别结果不变）。
  - **简单比喻：**就像用不同的“特征探测器”（刷子）在图片上巡逻，先找出边缘、颜色块等基础部件（卷积），然后把这些信息精简、汇总（池化），最后组合起来判断图片里是什么（比如是猫还是狗）。
- **循环神经网络 (Recurrent Neural Network, RNN)：**
  - **主要用途：**专门处理像句子、语音、股票价格这样的序列数据（数据点一个接一个，顺序很重要）。
  - **核心思想：**它有一个“记忆盒”。处理当前数据时，会结合“记忆盒”里之前存的信息，然后把新产生的信息更新到“记忆盒”里，再传给下一步。这样，理论上它能记住前面所有的历史信息。
  - **致命问题：**

    - **梯度消失/爆炸**：当序列很长时（比如很长的句子），RNN 在反向传播学习时，早期步骤的信息影响会变得极其微弱（消失）或异常巨大（爆炸），导致它实际上记不住太远的事情。
  - **改进版**：
    - **LSTM (长短期记忆网络)** 和 **GRU (门控循环单元)**：可以理解为“更聪明的记忆盒”。它们增加了“门”控机制（像开关一样），能自主决定记住什么、忘记什么、输出什么，从而更有效地学习长距离的依赖关系。
  - **简单比喻**：就像一条装配线，每个工位（RNN 单元）加工一个零件（序列中的一个数据），并把半成品（隐藏状态）传给下一个工位。LSTM/GRU 相当于给每个工位配了个智能助手，能判断哪些半成品需要重点保留，哪些可以丢弃。
- **Transformer (突破性架构)**：
  - **核心理念**：完全抛弃了 RNN 的“顺序加工”和 CNN 的“局部扫描”，转而采用“注意力”机制，让模型能够同时关注输入序列中的所有部分，并动态决定哪些部分之间的关系更重要。
  - **核心优势**：
    - **并行化**：RNN 必须一个一个按顺序处理单词，速度慢。Transformer 可以同时处理一句话里所有的单词，就像工厂从“流水线”变成“并行生产线”，训练速度极快。
    - **长距离依赖**：无论两个词在句子中隔得多远（比如“它”指代的是句首的某个名词），Transformer 都能通过注意力机制直接建立联系，没有距离衰减问题。
    - **可扩展性**：结构规整，像乐高积木一样，可以轻松堆叠很多层（加深）或增加每层的“宽度”（增加神经元数量），从而构建超大规模的模型。
  - **关键结构**：主要由两部分堆叠而成
    - **编码器**：负责“阅读理解”，把输入文本（比如一个句子）转换成一系列蕴含丰富信息的向量表示。
    - **解码器**：负责“写作生成”，根据编码器的输出和已经生成的部分，预测下一个词是什么。
    - **注意**：像 BERT 只用编码器，擅长理解任务；像 GPT 只用解码器，擅长生成任务；原始 Transformer 同时使用两者，擅长翻译这类“理解后再生成”的任务。
  - **划时代意义**：Transformer 是大语言模型（如 ChatGPT 背后的 GPT 系列，以及 BERT）的绝对基石。它的出现，使得训练包含数千亿参数的巨型模型成为可能，直接引爆了当前的 AI 革命。
  - **简单比喻**：想象你要理解一句话。Transformer 的做法不是从左到右读，而是把整句话铺开，然后让每个词都去“看”一遍句子中的所有其他词，并给自己和它们的关系打分（注意力分数）。这样，“苹果”这个词就能同时关联到“吃”、“红红的”、“很甜”，瞬间理解了全局上下文。

1.2.3 生成式人工智能：从理解到创造

生成式 AI 的目标是学习训练数据的分布 $p_{data}(x)$，并能够从中采样，生成新的、类似的数据样本 $x_{new} \sim p_{model}(x)$。这比仅仅判别数据类别（判别式模型）更具挑战性。

- **核心任务**：生成式 AI 旨在让模型创造出全新的、有意义的数字内容，而不是仅仅进行分类或预测。
  - **文本生成**：模型根据提示或上下文，续写、创作或总结文本。
    - **例如**：给 AI 输入“从前，有一只生活在森林里的小狐狸”，它可能会生成一个完整的故事。
  - **图像生成**：模型根据一段文字描述（提示词），生成一幅全新的图像。
    - **例如**：输入“一只穿着宇航服在月球上骑自行车的柯基犬”，生成对应的逼真或艺术图片。
  - **音频合成**：模型生成语音、音乐或其他声音。
    - **例如**：将文字转换成听起来非常自然的语音（TTS），或创作一段新的旋律。
  - **视频生成**：根据文字或图像提示，生成一段连贯的动态视频。
    - **例如**：输入“一只蝴蝶在花丛中飞舞的慢镜头”，生成几秒钟的视频片段。
  - **代码生成**：根据自然语言描述，自动生成可运行的计算机代码。
    - **例如**：告诉 AI“写一个 Python 函数，计算斐波那契数列”，它就会输出相应的代码。
  - **数据增强**：为已有数据集创造新的、合理的变体样本，用于扩充训练数据。
    - **例如**：有一张猫的图片，生成它的不同角度、光照下的新图片，帮助图像识别模型学得更好。
- **代表性技术**：以下是实现上述创造能力的几种主流“引擎”。
  - **生成对抗网络 (GANs)**
    - **核心思想**：设定两个神经网络互相“对抗”和“学习”。
    - **工作原理**：
      1. **生成器**：像一个“伪造者”，目标是生成尽可能逼真的假数据（如图片）。
      2. **判别器**：像一个“鉴定专家”，目标是准确判断输入数据是来自真实数据集还是生成器造的假。
    - **训练过程**：两者不断博弈。生成器努力骗过判别器，判别器努力不被骗。最终，生成器变得极其强大，能生成以假乱真的内容。
    - **类比**：就像学生学习（生成器）和老师考试（判别器）的对抗。老师不断出更难的题来抓住学生的知识漏洞（假数据），学生为了通过考试不得不学得越来越扎实，最终成为专家。
    - **典型应用**：早期的人脸生成、图像风格迁移。
  - **变分自编码器 (VAEs)**
    - **核心思想**：学习数据的“本质压缩包”（潜在空间），并从这个压缩包中解码生成新数据。

* **工作原理：**
  1. **编码器：**将输入数据（如图片）压缩成一个概率分布（均值和方差），表示在潜在空间中的位置。
  2. **采样：**从这个分布中随机采样一个点。
  3. **解码器：**将这个点“解压”回原始数据空间，试图重建或生成类似的新数据。
* **关键优势：**潜在空间是连续的，因此可以在两个特征之间平滑插值，生成中间态数据。
* **类比：**像学画画。先观察很多猫和狗的图片（编码），在心里形成一个关于“猫性”和“狗性”的抽象概念空间。然后，你可以在这个概念空间里，取一个 70% 像猫、30% 像狗的点，画出一个全新的“猫狗混合体”。
* **典型应用：**生成可控的面部表情、分子结构设计。

– **自回归模型**

* **核心思想：**将生成过程视为一个“顺序决策”过程。一次生成一个元素（如一个词、一个像素），每次生成都依赖于之前已经生成的所有内容。
* **工作原理：**模型就像一个“接龙大师”。给定一个开头（或空），它预测下一个最可能的词是什么，然后将这个词加到序列末尾，再基于新的序列预测下一个词，如此重复，直到生成完整内容。
* **数学表示：**$P(\text{完整序列}) = P(x_1) \times P(x_2|x_1) \times P(x_3|x_1, x_2) \times ...$
* **类比：**就像你写文章时，先写第一个词，然后根据已经写好的部分，思考下一个词写什么最合适，直到写完一段话。
* **典型代表:**GPT 系列就是最著名的自回归语言模型。它的“G”代表“Generative 生成式”，“P”代表“Pre-trained 预训练”，“T”就是 Transformer 架构。
* **特点：**擅长生成连贯的文本，是当前大语言模型的基石。

– **扩散模型**

* **核心思想：**灵感来源于热力学。学习将一个简单分布（如随机噪声）逐步“去噪”，转化为复杂数据分布（如图像）的过程。
* **工作原理（两步）：**
  1. **前向过程（加噪）：**对一张真实图片逐步添加高斯噪声，经过很多步后，图片会变成完全随机的噪声。
  2. **反向过程（去噪）：**模型学习这个加噪过程的逆过程——即，如何从一张纯噪声图片开始，一步步去除噪声，最终还原成一张有意义的图片。
* **生成过程:**要生成一张新图片，就从纯噪声开始，让训练好的模型反复进行“去噪”操作，最终得到一张清晰的图片。
* **类比：**就像一块大理石（纯噪声）。雕刻家（扩散模型）心里知道一座完美的雕像应该是什么样子，他通过不断凿掉多余的石料（去噪），最终将大理石变成雕像（目标图像）。

* **为何强大**：训练稳定，生成质量极高，细节丰富。
* **典型代表**：DALL-E 系列、Stable Diffusion、Midjourney 等主流文生图 AI 的核心技术。

1.3 基础模型与大模型：新范式的确立

2021 年，斯坦福大学《On the Opportunities and Risks of Foundation Models》报告正式提出“基础模型”概念，标志着 AI 进入新范式。

1.3.1 定义与核心特征

基础模型：在广泛数据（通常是通过自监督学习）上训练的大规模模型，可以适应（例如，通过微调）到广泛的下游任务。

1. **规模性**：参数量巨大（十亿至万亿级别），训练数据海量（千亿至万亿 token 级别），计算消耗惊人（千卡/万卡 GPU 集群训练数月）。
2. **预训练与自监督学习**：在大规模无标签数据上，通过设计“前置任务”（如掩码语言建模、下一词预测）进行预训练，让模型学习通用的世界知识和表示。
3. **涌现与同质化**：
   - **涌现能力**：当模型规模超过某个阈值，性能会出现不连续的、急剧的提升，并产生小模型没有的能力（如多步推理、指令理解）。
   - **同质化**：同一基础模型架构（如 Transformer）通过微调可应用于无数不同任务，减少了为每个任务从头设计模型的需要。
4. **适应性**：通过提示工程（Prompting）、上下文学习（In-context Learning）、微调（Fine-tuning）等方式，可快速适配到具体应用。

1.3.2 里程碑与发展历程

大语言模型的发展经历了多个关键里程碑，每一个都代表了技术的重大突破：

- **2017 年：Transformer** 架构被提出，其完全基于自注意力机制的创新设计，彻底改变了序列建模的方式，成为所有现代大模型的基础架构。
- **2018 年**：OpenAI 的 **GPT-1** 展示了生成式预训练的潜力；Google 的 **BERT** 则开创了双向编码器预训练范式，在众多 NLP 任务上取得突破性成果。
- **2020 年：GPT-3**（175B 参数）发布，首次大规模验证了“缩放定律”，并展示出强大的少样本/零样本学习能力，引发行业广泛关注。
- **2021 年**：多模态预训练模型兴起，OpenAI 的 **CLIP** 和 **DALL-E** 等模型成功连接了文本与图像模态，开启了多模态 AI 的新篇章。
- **2022 年 11 月**：OpenAI 发布 **ChatGPT**（基于 GPT-3.5/4），这是一个基于 RLHF 优化的对话模型。其惊艳的对话、推理和指令遵循能力瞬间引爆全球，真正开启了 AI 大

模型应用的新纪元。

- **2023 年**：开源模型蓬勃发展，Meta 的 **LLaMA**、阿里的**通义千问**、百度的**文心一言**等众多模型相继发布，形成“”百模大战“”局面，极大地推动了技术的民主化和应用落地。
- **2024 年**：**GPT-4o**、**Claude 3**、**Gemini 1.5** 等模型发布，多模态能力进一步增强，上下文窗口扩展至百万 token 级别，模型的推理、理解和生成能力持续突破。

1.3.3 大模型的分类谱系

大模型可以根据其处理的主要数据模态进行分类，主要包括以下几种类型：

- **大语言模型 (LLM)**：专门处理和理解自然语言文本的模型。它们通常基于 Transformer 架构（特别是解码器架构），经过大规模文本数据训练，能够进行文本生成、翻译、摘要、问答、代码生成、逻辑推理和角色扮演等任务。代表性模型包括 GPT-4、Claude、LLaMA 系列、通义千问、文心一言等。
- **多模态大模型**：能够同时处理和关联多种类型输入（如文本、图像、音频、视频）的模型。其关键技术包括：
  - **对齐 (Alignment)**：将不同模态的数据映射到统一的语义空间，例如 CLIP 模型将图像和文本对齐到同一向量空间。
  - **融合 (Fusion)**：在模型内部设计专门的架构来融合多模态信息，例如 Flamingo、BLIP-2 等模型。

  这类模型的应用涵盖视觉问答、图像描述生成、文生图、文生视频以及具身智能等领域。代表性模型有 GPT-4V（视觉理解）、Gemini、DALL-E 3（文生图）、Sora（文生视频）等。
- **视觉基础模型**：主要专注于图像或视频的理解与生成任务。可以视为多模态模型的一个特例（专注于视觉模态）。代表性模型包括 Vision Transformer (ViT)、Stable Diffusion（文生图）、Segment Anything Model (SAM，图像分割) 等。
- **音频处理大模型**：专注于语音和音频的理解与生成。代表性模型有 Whisper（语音识别）、AudioLM（音频生成）等。

1.4 大模型的训练：从通用到对齐

训练一个可用的大模型是一个复杂的系统工程，通常分为三个阶段。

1.4.1 第一阶段：大规模预训练 (Pre-training)

- **目标**：从海量无标签数据中学习通用的语言表示和世界知识。
- **数据**：互联网文本、书籍、代码、学术论文等，总量可达数十 TB。
- **任务**：自监督学习任务。

  - **自回归语言建模 (因果语言建模, CLM)**：模型被训练来预测下一个 token（词元），给定之前的所有 tokens，预测下一个 token。$\mathcal{L}_{CLM} = -\sum_t \log P(x_t|x_{<t};\theta)$。这是 GPT 系列采用的方法。GPT 通过这种方式学会了语言的规律，从而能够生成连贯、合理的文本。（自回归语言建模就像是一个猜下一个词的游戏）
  - **掩码语言建模 (MLM)**：随机掩盖部分输入 token，让模型预测被掩盖的词。$\mathcal{L}_{MLM} = -\sum_{i\in M} \log P(x_i|x_{\setminus M};\theta)$。这是 BERT 采用的方法。掩码语言建模是 BERT 的核心预训练机制，通过双向上下文预测被掩盖词，使模型获得丰富的语言理解能力。
- **产出**："基座模型" (Base Model)。它知识渊博，但不擅长遵循人类指令，且可能生成有害内容。

1.4.2 第二阶段：有监督微调 (Supervised Fine-Tuning, SFT)

- **目标**：教会模型理解并遵循人类的指令。
- **数据**：高质量的人类标注对话或指令数据，格式为 $\{(instruction, input), output\}$。
- **方法**：在预训练模型的基础上，使用指令数据继续训练，通常采用与预训练相同的语言建模损失函数。
- **产出**："SFT 模型"。模型学会了"对话格式"和基本的指令遵循，但可能还不够"听话"或"有用"。

1.4.3 第三阶段：基于人类反馈的强化学习 (RLHF)

RLHF 是使 ChatGPT 等模型输出更符合人类偏好的关键技术，通常分为三步：

1. **收集人类偏好数据**：让标注员对同一个提示词下的多个模型回复进行排序，产生偏好对 (y_w, y_l)，其中 y_w 是更优回复。
2. **训练奖励模型 (Reward Model, RM)**：用一个较小的模型（如 6B 参数）来学习人类的偏好。其目标是，对于给定的提示 x 和回复 y，输出一个标量奖励 $r_\theta(x, y)$。训练时，让奖励模型对偏好对 (y_w, y_l) 的输出差值最大化：

$$\mathcal{L}_{RM} = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))\right]$$

   其中 σ 是 sigmoid 函数。
3. **强化学习优化策略模型**：将 SFT 后的模型作为初始策略 π^{SFT}。在强化学习框架下，目标是优化策略 π_ϕ 以最大化奖励模型的输出，同时防止其偏离初始 SFT 模型太远（使用 KL 散度作为惩罚项）：

$$\max_{\pi_\phi} \mathbb{E}_{x\sim D, y\sim\pi_\phi(\cdot|x)}\left[r_\theta(x, y) - \beta \log \frac{\pi_\phi(y|x)}{\pi^{SFT}(y|x)}\right]$$

   通常使用 PPO（近端策略优化）算法来求解。这个过程鼓励模型生成高奖励的回复，同时保持语言的流畅性和多样性。

1.5 大模型的内部工作机制

1.5.1 从文本到 Token：分词化

分词是将原始文本转换为模型可处理数字序列的第一步。

- **子词分词算法 (Subword Tokenization)**：主流方法。
  - **Byte Pair Encoding (BPE)**: GPT 系列使用。从字符开始，迭代合并出现频率最高的相邻符号对。
  - **WordPiece**: BERT 使用。与 BPE 类似，但合并依据是可能性（似然）而非频率。
  - **SentencePiece**: 将文本视为 Unicode 字符序列，无需预分词，支持多语言。
- **词表 (Vocabulary)**：分词后所有唯一 Token 的集合。大小通常在数万到数十万。每个 Token 对应一个唯一的整数 ID。

1.5.2 Transformer 解码器的自回归生成

以 GPT 为代表的解码器模型，其文本生成是一个自回归过程。

1. **输入表示**：提示文本被分词为 Token IDs 序列 $[t_1, t_2, ..., t_n]$。每个 Token ID 被转换为词嵌入向量，并加上位置编码（Positional Encoding）以注入顺序信息。
2. **堆叠 Transformer 解码器块**：
   - **掩码自注意力层 (Masked Self-Attention)**：每个位置的 Token 只能关注它之前（包括自身）的位置，确保生成是因果的。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \tag{1.2}$$

     其中 M 是一个上三角为 $-\infty$ 的掩码矩阵，使得未来位置的信息被屏蔽。
   - **前馈神经网络层 (FFN)**：对每个位置的表示进行非线性变换。
   - **残差连接与层归一化**：缓解深层网络训练中的梯度消失问题。
3. **输出层**：最后一个解码器块的输出，经过一个线性层（词表大小维度）和 softmax 函数，得到下一个 Token 在整个词表上的概率分布 $P(t_{n+1}|t_1, ..., t_n)$。
4. **采样策略**：
   - **贪婪搜索 (Greedy Search)**：选择概率最高的 Token。简单但容易导致重复、乏味的输出。
   - **集束搜索 (Beam Search)**：维护 k 个最有可能的序列候选。质量通常更好，但可能缺乏多样性。
   - **核采样 (Top-p Sampling)**：从累积概率超过阈值 p 的最小 Token 集合中随机采样。在多样性和质量间取得较好平衡。
   - **温度采样 (Temperature Sampling)**：在 softmax 前将 logits 除以温度参数 T。T 高则分布平滑，输出随机；T 低则分布尖锐，输出确定。

5. **自回归循环**：将上一步采样得到的 Token ID 追加到输入序列末尾，重复步骤 1-4，直到生成结束符或达到最大长度。

1.6 高效应用大模型：提示工程与 API 集成

1.6.1 提示工程

提示是与大模型交互的主要界面。好的提示能极大激发模型潜力。

核心原则

- **清晰与具体**：明确任务、角色、格式、长度等要求。
- **提供上下文与示例 (Few-shot Prompting)**：在提示中给出少量输入-输出示例，引导模型理解任务格式。
- **系统指令 (System Prompt)**：设定模型的角色、行为准则和回答风格。通常放在对话的开头。

高级技巧

- **思维链 (Chain-of-Thought, CoT)**：对于复杂推理问题，在提示中要求或示例模型“逐步思考”。

```
问题：小明有5个苹果，他给了小红2个，又买了3个，现在他有几个苹果？
让我们一步步思考：
1. 一开始有5个苹果。
2. 给小红2个后，剩下 5 - 2 = 3个苹果。
3. 再买3个后，现在有 3 + 3 = 6个苹果。
所以，小明现在有6个苹果。
```

- **生成-验证-修订 (Generate-Verify-Revise)**：让模型先生成一个答案，然后自我检查，最后修订错误。
- **角色扮演 (Persona)**：赋予模型一个特定角色（如资深律师、幽默的脱口秀演员），以获得特定风格的回复。

1.6.2 程序化调用：API 与批量处理

企业级应用通常通过 API 将大模型能力集成到工作流中。

1.7 超越原生能力：RAG 与插件系统

1.7.1 RAG：为模型注入动态知识

深入理解 RAG 工作流

RAG（检索增强生成）系统的核心思想是将外部知识库的信息动态地注入到大模型的生成过程中，以弥补大模型自身知识的局限性（如非实时、缺乏私有知识）。其工作流程可清晰地分为两个主要阶段：**索引阶段（Indexing）**和 **检索与生成阶段（Retrieval & Generation）**。

第一阶段：索引阶段：此阶段的目标是为外部知识库（如企业内部文档、产品手册、最新研究报告等）构建一个高效可检索的索引，通常是一个向量数据库。

1. **数据加载与解析：**从各种来源（PDF、HTML、Word、数据库、API 等）加载原始文档，并解析提取出纯文本内容。这一步的挑战在于准确处理不同格式（如 PDF 中的表格、图表）和编码。
2. **文本分割（Chunking）：**将长文档分割成大小适宜的文本块（Chunks）。这是 RAG 系统的关键步骤之一，分割策略需在保留语义完整性和控制块长度之间取得平衡。常见的策略包括：
   - **固定大小分割：**简单但可能割裂完整句子。
   - **按句子/段落分割：**保留自然语义边界，但长度可能不均。
   - **重叠滑动窗口：**保持块间上下文连贯，但增加存储和计算开销，可能引入冗余。
   - **基于语义的分割：**理论上最优，保证块内语义完整，但实现复杂，依赖语义分割模型，计算成本高。
   - **递归字符分割**（如 LangChain 的 RecursiveCharacterTextSplitter）：平衡语义和长度，先按大分隔符（如双换行）分割，再递归细分，通用性强，效果相对稳定。
3. **向量化（Embedding）：**使用嵌入模型（如 OpenAI 的 text-embedding-ada-002）将每个文本块转换为一个高维向量（例如 1536 维）。这个向量捕捉了文本的语义信息，语义相似的文本其向量在空间中的距离也较近。
4. **存储索引：**将文本块、其对应的向量以及可能的元数据（如来源、创建时间）存储到向量数据库（如 Chroma, Pinecone, Milvus）中，并建立高效的索引以支持快速相似性搜索。

第二阶段：检索与生成阶段：此阶段响应用户查询，并生成结合了外部知识的答案。

1. **查询向量化：**当用户提出一个问题（Query）时，使用与索引阶段相同的嵌入模型将这个问题也转换为一个向量。
2. **语义检索：**在向量数据库中，计算查询向量与所有存储的文档块向量之间的相似度（常用余弦相似度）。根据相似度分数排序，返回最相似的 Top-K 个文档块。这个过程通常非常快速（毫秒级）。
3. **构建上下文：**将检索到的 Top-K 个文档块按相关性排序，合并成一个长的上下文字符串。有时会进行后处理，如去重、重排序（使用更精细但更慢的交叉编码器模型）。

4. **提示工程与构造**：将用户查询和检索到的上下文按照预定义的模板组合成一个新的提示（Prompt）。模板通常指示模型基于给定的上下文来回答问题，并可能包含指令来避免模型胡编乱造（如 “” 如果上下文没有相关信息，请回答’ 我不知道’ “”）。
5. **调用大模型生成**：将构造好的提示输入大语言模型（如 GPT-4）。模型基于这个包含外部知识的提示，生成最终答案。
6. **返回答案**：将模型生成的答案返回给用户。有时答案中还会附上引用的源文档片段，以增强可信度。

整个 RAG 流程就像是让大模型进行一场 “开卷考试”：索引阶段是准备 “参考书”（向量化知识库），检索阶段是根据 “考题”（用户查询）从 “参考书” 中快速找到相关 “段落”（文档块），生成阶段是结合找到的 “段落” 和自己的理解来 “答题”。

向量检索与相似度计算

检索的核心是计算查询向量与文档块向量的相似度。常见的相似度度量包括：

- **余弦相似度 (Cosine Similarity)**：最常用，衡量向量方向的相似性，与长度无关。

$$\text{sim}_{\cos}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\|\|\mathbf{d}\|} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \sqrt{\sum_{i=1}^{n} d_i^2}}$$

- **点积 (Dot Product)**：受向量长度影响，长度越大点积可能越大。有时在向量归一化后使用。
- **欧几里得距离 (Euclidean Distance)**：衡量向量空间中的直线距离。距离越小越相似。

$$d_{\text{euclid}}(\mathbf{q}, \mathbf{d}) = \sqrt{\sum_{i=1}^{n} (q_i - d_i)^2}$$

向量数据库的高级特性

现代向量数据库不仅仅是存储和检索向量，还提供：

- **混合搜索 (Hybrid Search)**：结合向量语义搜索和传统的关键字搜索（如 BM25），提升召回率和准确性。
- **元数据过滤**：在向量搜索的基础上，根据文档的元数据（如日期、作者、类别）进行过滤。
- **动态数据管理**：支持增量更新和删除，以及数据分片和分布式存储以应对海量数据。

1.7.2 RAG 的挑战与系统化优化

索引阶段的挑战

- **文档解析准确性**：复杂 PDF（含表格、图表、双栏）、扫描件（需 OCR）、网页（动态内容）的准确提取是首要难题。

- **分块策略的权衡：**
  文本分块策略各有优缺点，需要根据实际场景选择：
  - **固定大小分块：**实现简单，易于管理，但可能割裂完整语义单元（如句子中间）。
  - **按句子/段落分割：**保留自然语义边界，但句子/段落长度不一，可能超出上下文窗口或太短。
  - **重叠滑动窗口：**保持块间上下文连贯，但增加存储和计算开销，可能引入冗余。
  - **基于语义的分割：**理论上最优，保证块内语义完整，但实现复杂，依赖语义分割模型，计算成本高。
  - **递归字符分割 (LangChain)：**平衡语义和长度，先按大分隔符（如双换行）分，再递归细分，通用性强，效果相对稳定。

查询阶段的挑战与优化

- **查询改写/扩展 (Query Transformation/Expansion)：**原始查询可能不够精确。可以：
  - 使用大模型将查询改写得更清晰、具体。
  - 基于查询生成相关问题（HyDE - Hypothetical Document Embeddings）。
  - 对复杂查询进行分解（Step-back Prompting, Decompose Query）。
- **重排序 (Re-ranking)：**初步向量检索（高召回率，Top-K 可能较大）得到的文档块，使用一个更精确但更慢的 “” 交叉编码器 “” 模型进行精排，选取最相关的 Top-N 个。

Listing 1.1: 使用交叉编码器重排序

```
# 伪代码：初步检索 + 重排序
from sentence_transformers import CrossEncoder

# 1. 初步向量检索（召回）
retrieved_chunks = vector_db.similarity_search(query, k=20)

# 2. 使用交叉编码器重排序
cross_encoder = CrossEncoder('cross-encoder/ms-marco-MiniLM-L-6-v2')
pairs = [[query, chunk.text] for chunk in retrieved_chunks]
scores = cross_encoder.predict(pairs)

# 3. 根据得分排序并选取Top-5
ranked_indices = np.argsort(scores)[::-1][:5]
final_contexts = [retrieved_chunks[i] for i in ranked_indices]
```

- **上下文管理/压缩：**检索到的上下文可能很长，超过模型限制或包含噪声。
  - **选择性上下文：**只提取与查询最相关的句子。

  - **摘要式压缩**：用大模型对长上下文进行摘要，再将摘要作为提示的一部分。
- **迭代检索与问答**：对于需要多步推理的复杂问题，可以采用“”检索-阅读-再检索“”的多轮迭代方式。

1.7.3 插件与工具调用

大模型作为“大脑”，可以通过插件（工具）来获取实时信息或执行具体操作。

- **工作原理 (ReAct 模式)**：
  1. 用户提问。
  2. 模型思考是否需要调用工具，以及调用哪个工具、传入什么参数。
  3. 模型生成一个结构化的工具调用请求（如 JSON）。
  4. 应用程序执行工具，并返回结果。
  5. 模型将工具返回的结果整合到上下文中，生成最终回复给用户。
- **工具描述**：需要以结构化的方式（如 OpenAI 的 Function Calling 格式）向模型描述工具的功能、输入参数和返回值。

1.8 LangChain：大模型应用开发框架

LangChain 是一个用于构建基于大语言模型应用的框架，其核心设计理念是链式组合。

1.8.1 核心抽象概念

- **组件 (Components)**：提供与各种外部系统交互的模块化工具，如模型 I/O、数据检索、记忆、工具等。
- **链 (Chains)**：将组件（或其他链）按特定顺序组合起来，以完成复杂任务。链是可组合的。
- **代理 (Agents)**：由大模型驱动的“自动决策者”。模型根据目标，动态决定调用哪些工具以及调用顺序。
- **记忆 (Memory)**：在链或代理的多次调用之间持久化状态（如对话历史）。

1.8.2 LangChain 表达式语言 (LCEL)

LCEL 提供了一种声明式、链式的方式来组合组件，是构建复杂应用的首选方式。

Listing 1.2: 使用 LCEL 构建复杂链

```
from langchain_core.prompts import ChatPromptTemplate
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser,
    JsonOutputParser
```

```
from langchain_core.runnables import RunnablePassthrough, RunnableLambda

# 定义多个组件
prompt1 = ChatPromptTemplate.from_template("将以下中文翻译成英文: {input}")
prompt2 = ChatPromptTemplate.from_template("总结以下英文文本: {text}")
llm = ChatOpenAI(model="gpt-3.5-turbo")
parser = StrOutputParser()

# 使用 LCEL 的管道操作符 `|` 组合链
# 链1: 翻译
translation_chain = prompt1 | llm | parser
# 链2: 总结
summarization_chain = prompt2 | llm | parser
# 组合链: 先翻译, 再总结
combined_chain = {"text": translation_chain} | summarization_chain

# 调用组合链
result = combined_chain.invoke({"input": "今天天气真好, 适合去公园散步。"})
print(result) # 输出英文摘要
```

1.8.3 使用 LangChain 构建高级 RAG 应用

LangChain 极大地简化了 RAG 应用的构建。

Listing 1.3: 带重排序和元数据过滤的 RAG 链

```
from langchain_community.vectorstores import Chroma
from langchain_openai import OpenAIEmbeddings, ChatOpenAI
from langchain.retrievers import ContextualCompressionRetriever
from langchain.retrievers.document_compressors import LLMChainExtractor
from langchain.retrievers import EnsembleRetriever
from langchain_community.retrievers import BM25Retriever
from langchain_core.documents import Document

# 1. 准备向量检索器和关键词检索器
embeddings = OpenAIEmbeddings()
vectorstore = Chroma(persist_directory="./chroma_db",
    embedding_function=embeddings)
```

```
vector_retriever = vectorstore.as_retriever(search_kwargs={“”k“”: 10})

# 假设我们有文本列表 `texts` 和对应的 `metadatas`
bm25_retriever = BM25Retriever.from_texts(texts, metadatas=metadatas)
bm25_retriever.k = 10

# 2. 创建混合检索器 (Ensemble Retriever)
ensemble_retriever = EnsembleRetriever(
    retrievers=[vector_retriever, bm25_retriever],
    weights=[0.7, 0.3] # 权重可调
)

# 3. 创建上下文压缩器（使用LLM提取最相关部分）
llm = ChatOpenAI(temperature=0)
compressor = LLMChainExtractor.from_llm(llm)
compression_retriever = ContextualCompressionRetriever(
    base_compressor=compressor,
    base_retriever=ensemble_retriever
)

# 4. 定义RAG链
from langchain import hub
prompt = hub.pull(“rlm/rag-prompt”) # 拉取社区优质提示模板
llm = ChatOpenAI(model_name=“”gpt-4“”, temperature=0)

def format_docs(docs):
    return “\n\n”.join(doc.page_content for doc in docs)

rag_chain = (
    {“”context“”: compression_retriever | format_docs, “”question“”: RunnablePassthrough()}
    | prompt
    | llm
    | StrOutputParser()
)

# 5. 提问
answer = rag_chain.invoke(“什么是Transformer模型？”)
print(answer)
```

1.8.4 LangChain 生态与生产化工具

- **LangSmith**：用于调试、测试、评估和监控 LangChain 应用的平台。可以跟踪链的每一步输入输出，分析延迟和成本，进行回归测试。
- **LangGraph**：用于构建有状态、多参与者的复杂代理工作流。支持循环、条件分支，非常适合构建需要规划和执行多步任务的自主智能体。
- **LangServe**：轻松将任何 LangChain 链部署为 REST API。

1.9 大模型微调：从通用到专属

当提示工程和 RAG 无法满足需求时（如需要模型掌握特定领域知识、遵循独特风格格式），微调是更深入的解决方案。

1.9.1 微调 vs. RAG：深度对比

微调与 RAG 是两种不同的知识注入方式，各有优劣：

- **核心机制**：
  - **微调 (Fine-tuning)**：用领域数据更新模型内部参数，将知识“内化”到模型权重中。
  - **RAG (检索增强生成)**：从外部知识库检索信息并注入上下文，知识存储在外部。
- **知识存储**：
  - **微调**：知识以参数化的方式存储在模型权重中。
  - **RAG**：知识以非参数化的方式存储在外部的向量库或数据库中。
- **知识更新**：
  - **微调**：困难且昂贵，需要重新训练或持续训练，成本高。
  - **RAG**：容易且廉价，只需更新外部知识库，无需修改模型。
- **可解释性**：
  - **微调**：较低，知识被编码到权重中，难以追溯来源。
  - **RAG**：较高，可以追溯到检索到的源文档，便于验证。
- **幻觉控制**：
  - **微调**：可能内化错误，仍会产生幻觉。
  - **RAG**：通过引用源文档，可减少幻觉，易于验证答案准确性。
- **计算开销**：
  - **微调**：训练成本高，但推理速度快（无需额外检索）。
  - **RAG**：训练成本低，但推理延迟高（需要检索和整合外部信息）。
- **适用场景**：

  - **微调**：适用于任务/风格固化、对推理速度要求高、领域术语复杂的场景。
  - **RAG**：适用于知识频繁更新、处理大量私有文档、需要溯源和减少幻觉的场景。

1.9.2 主流微调方法

- **全参数微调 (Full Fine-tuning)**：更新模型的所有参数。效果最好，但计算和存储成本最高。
- **参数高效微调 (Parameter-Efficient Fine-Tuning, PEFT)**：只更新一小部分额外的参数，大部分预训练参数冻结。在效果接近全微调的同时大幅降低成本。
  - **LoRA (Low-Rank Adaptation)**：在模型的注意力层中注入低秩矩阵，只训练这些新增的小矩阵。
  - **Adapter**：在 Transformer 层中插入小型神经网络模块，只训练这些适配器。
  - **前缀微调 (Prefix-Tuning)**：在输入序列前添加可训练的任务特定前缀向量。

1.9.3 微调实践流程

1. **数据准备**：收集高质量的指令-输出对数据。格式需统一（如 Alpaca 格式）。通常需要数千到数万条。
2. **环境配置**：准备 GPU 资源，安装深度学习框架（如 PyTorch）和微调库（如 Hugging Face Transformers, PEFT）。
3. **模型与训练配置**：选择基座模型，确定微调方法（如 LoRA），设置超参数（学习率、批大小、训练轮数）。
4. **训练与评估**：在训练集上微调，在验证集上监控损失和指标（如准确率、BLEU）。
5. **推理部署**：将训练好的适配器权重与基础模型合并，或单独加载，进行推理服务。

1.10 未来趋势与挑战

1.10.1 技术趋势

- **模型规模与效率的平衡**：研究方向从单纯追求“更大”转向“更聪明”和“更高效”。MoE（混合专家）模型、模型量化、蒸馏、剪枝等技术是热点。
- **多模态深度融合**：从简单的图文对齐走向真正统一的理解与生成模型，能处理任意模态组合的输入输出。
- **智能体 (Agent) 范式的普及**：大模型作为规划和控制核心，能够自主调用工具、与环境交互、完成复杂长周期任务的 AI 智能体将成为主流应用形态。
- **推理能力的长足进步**：通过思维链、思维树、程序辅助等提示技术，以及专门针对数学、代码、逻辑推理的模型训练，大模型的推理能力将持续突破。

- **开源与闭源的竞合**：强大的开源模型（如 Llama、Qwen）推动技术民主化，降低应用门槛；闭源模型则在尖端能力上保持领先。生态将更加多元。

1.10.2 主要挑战

- **安全与对齐**：如何确保大模型输出无害、诚实、符合人类价值观，避免偏见、歧视和滥用，是长期核心挑战。
- **成本与能耗**：大模型的训练和推理消耗巨量计算资源和电力，可持续发展是重要议题。
- **数据瓶颈与版权**：高质量训练数据逐渐枯竭，数据版权和隐私问题日益突出。
- **评估难题**：如何全面、客观地评估大模型在开放域任务上的能力，缺乏统一标准。
- **可靠性（“幻觉”）**：模型生成事实性错误内容的问题尚未根本解决，在关键领域应用仍需谨慎。

1.11 结语

AI 大模型的发展正在重塑我们与信息和技术互动的方式。从理论基础到训练方法，从核心机制（如 Transformer、自回归生成）到应用技术（提示工程、RAG、微调），再到开发框架（如 LangChain），构成了一个庞大而充满活力的技术体系。掌握这套体系，不仅需要理解其背后的数学和算法原理，更需要通过实践去探索如何将模型能力与具体业务场景相结合，解决实际问题。

第二章　基于大语言模型的增强型 Text-to-SQL 解析方法：SLENet

2.1　引言

2.1.1　研究背景与意义

Text-to-SQL 是自然语言处理领域的核心任务之一，其目标是将用户以自然语言提出的问题（例如“显示得分最高的提交结果”）转换为可在关系数据库上执行的标准 SQL 查询语句。该技术能够使不具备 SQL 专业知识的业务人员、数据分析师等非技术用户直接访问数据库，极大地促进了数据驱动的决策过程，在商业智能、数据分析和人机交互等领域具有广阔的应用前景。

2.1.2　技术挑战与发展历程

尽管 WikiSQL、Spider 等基准数据集的建立推动了该领域的快速发展，但构建一个在真实场景下表现鲁棒的 Text-to-SQL 系统仍面临诸多挑战：

- **自然语言的复杂性与歧义性**：同一种查询意图可以有多种不同的表达方式，且自然语言中存在共指、省略等现象。
- **数据库模式理解与链接**：模型需要准确理解数据库的表结构、列含义以及表间关系，并将自然语言中的概念正确链接到具体的数据库模式元素上。
- **复杂 SQL 结构的生成**：对于涉及多表连接、嵌套子查询、聚合函数、排序和分页等复杂操作的 SQL，生成语法正确且语义准确的查询具有很高难度。
- **跨领域泛化能力**：模型在训练阶段未见过的数据库模式上能否保持良好的性能是一个关键挑战。

为应对这些挑战，Text-to-SQL 技术经历了一系列演进：

1. **基于规则与模板的方法**：早期方法依赖于大量人工定义的规则，灵活性与泛化能力差。
2. **基于深度学习的方法**：利用序列到序列模型、SQLNet 等模型自动学习从自然语言到 SQL 的映射。
3. **基于预训练语言模型的方法**：采用 BERT、T5 等模型，通过微调利用其强大的语义表示能力。

4. **基于大语言模型的方法**：近期，如 GPT、LLaMA 等大语言模型通过其强大的上下文学习与代码生成能力，为该任务带来了新的突破。

2.1.3　本文工作：SLENet 模型概述

论文提出的 SLENet 模型是当前基于 LLM 范式的先进代表。

其核心思想是：**利用大语言模型作为强大的语义理解器，同时通过一个受严格语法约束的解码器来确保生成的 SQL 的结构正确性**。具体来说，SLENet 包含三大创新点：

1. **基于搜索的提示优化**：从外部数据集中检索与当前用户问题相似的示例，构建富含上下文的提示，以增强 LLM 对当前任务的理解。
2. **LLM 增强嵌入**：利用 LLaMA 等大语言模型处理增强后的提示、用户问题和数据库模式，生成高质量的语义嵌入表示。
3. **语法约束解码器**：设计一个基于 Transformer 的解码器，在解码的每一步都利用 SQL 的语法规则对生成的令牌进行约束，确保最终输出的 SQL 查询在语法上是正确的。

2.2　SLENet 方法详解

2.2.1　整体框架与问题形式化

SLENet 解决的任务可形式化定义为：给定一个自然语言问题序列 $X = \{x_1, x_2, \ldots, x_{|X|}\}$ 和一个数据库模式 S（包含表集合和列集合），目标是生成一个语法正确且能准确反映用户查询意图的 SQL 语句 Y_{sql}。

SLENet 的处理流程可分为三个主要阶段，其整体数据处理流程如下所示：

1. **输入增强**：通过搜索外部语料库，为输入问题 X 找到一组最相关的示例，构成提示 P。
2. **语义编码**：将增强后的输入 $I = \text{concat}(P, X, S)$ 送入大语言模型，得到：
   - 自然语言问题的增强嵌入表示 E_{NL}
   - 数据库模式的嵌入表示 E_S
   - 一个由 LLM 初步生成但可能包含错误的初始 SQL 查询 Y_{init}
3. **约束解码**：将 $E = \text{concat}(E_{NL}, E_S, Y_{init})$ 作为语法约束解码器的输入，逐步生成最终的 SQL 查询。

2.2.2　组件一：基于搜索的提示优化

该组件旨在为 LLM 提供更丰富的上下文信息，使其更好地理解当前任务。其具体步骤为：

1. **数据源**：使用外部 Text-to-SQL 数据集，如 WikiSQL。
2. **相似度计算**：计算用户问题 X 与外部数据集中所有问题 Q_i 的语义相似度：

$$\text{Relevance}(X, Q_i) = \text{Sim}(f(X), f(Q_i))$$

其中，$f(\cdot)$ 为文本编码函数，Sim 通常为余弦相似度。

3. **提示构建**：选取相似度最高的前 N 个 (Q_i, SQL_i) 对，与当前问题 X 和模式 S 拼接，构成最终输入：

$$I = [P_1; P_2; ...; P_N; X; S]$$

2.2.3　组件二：LLM 增强嵌入

此阶段利用 LLM 处理增强后的输入 I，生成高质量的语义表示。

- **骨干网络**：论文采用 LLaMA 7B 作为基础 LLM，因其在性能与计算开销间取得了良好平衡。
- **输出**：LLM 主要产生三部分输出：
  1. 问题与模式的令牌级嵌入：E_{NL} 和 E_{Schema}。
  2. 一个初始的 SQL 查询 $Y_{init} = \text{LLM}(I)$。需要注意的是，由于此步骤是 LLM 的自由生成，Y_{init} 可能包含语法或语义错误。

2.2.4　组件三：语法约束解码器

这是 SLENet 确保 SQL 语法正确性的核心。该解码器是一个 6 层的 Transformer 结构，其生成过程是自回归的，且每一步都受到 SQL 语法规则的约束。

解码过程形式化

解码过程可表示为：

$$p(Y|X, S) = \prod_{t=1}^{T} p(y_t|y_{<t}, E, \mathcal{C}_t)$$

其中，y_t 是在时间步 t 生成的令牌，$y_{<t}$ 是之前已生成的令牌序列，E 是编码器输出的嵌入表示，$\mathcal{C}_t$ 是根据当前已生成序列和 SQL 语法，在时间步 t 所有允许出现的有效令牌集合。

语法掩码机制

关键创新在于引入了语法掩码 M_t：

$$M_t[i] = \begin{cases} 0, & \text{如果令牌} i \in \mathcal{C}_t \\ -\infty, & \text{如果令牌} i \notin \mathcal{C}_t \end{cases}$$

在解码器输出 logits z_t 后，先加上掩码 M_t，再进行 softmax 操作，以抑制无效令牌的出现概率：

$$p(y_t|...) = \text{softmax}(z_t + M_t)$$

动作空间

解码过程可视为执行一系列动作来构建 SQL 的抽象语法树：

- **APPLYRULE**：应用 SQL 语法的一个产生式规则。
- **SELECTCOLUMN**：从模式中选择一个列。
- **SELECTTABLE**：从模式中选择一个表。

每一步允许的动作集合 $\mathcal{C}_t$ 由当前已构建的语法树状态和 SQL 语法共同决定。

2.3　实验评估与分析

2.3.1　实验设置

- **数据集**：在 **Spider** 数据集上进行评估。该数据集是复杂的跨领域 Text-to-SQL 基准，包含：
  - 10,181 条问题-SQL 对。
  - 200 多个数据库，覆盖 138 个不同领域。
  - 将查询按复杂度分为：简单、中等、困难、极难四个等级。
- **评估指标**：主要使用**精确匹配准确率**，即模型生成的 SQL 与标准答案在字符串层面完全一致的样本比例。
- **基线模型**：与多种先进方法进行比较，包括：
  - **Seq2Seq**: 基础的序列到序列模型。
  - **SQLNet**: 基于草图（sketch）的槽填充方法。
  - **IRNet**: 使用中间表示来桥接语义鸿沟。
  - **PICARD**: 采用约束解码确保语法正确性。
  - **SmBoP**: 基于半自回归 bottom-up 解析的方法。

2.3.2　主要结果

表 2.1: SLENet 与基线模型在 Spider 数据集上的精确匹配准确率对比

模型	开发集	测试集
Seq2Seq	1.9%	3.7%
SQLNet	10.9%	12.4%
IRNet	61.9%	54.7%
PICARD	69.7%	65.6%
SmBoP	-	71.1%
SLENet	**75.2%**	**74.1%**

结果分析：

- SLENet 在开发集和测试集上均达到了最优性能，测试集准确率为 74.1%。
- 与次优的 SmBoP（71.1%）相比，SLENet 有 3 个百分点的显著提升。
- 这证明了将 LLM 的语义能力与语法约束解码相结合的有效性。

2.3.3　消融实验

为了验证各组件的重要性，论文进行了深入的消融研究。

LLM 增强嵌入的有效性

表2.2展示了为不同基线模型引入 LLM 增强嵌入后的性能变化。

表 2.2: LLM 增强嵌入对不同模型性能的提升（测试集准确率）

模型	无 LLM 增强	有 LLM 增强	提升幅度
Seq2Seq	3.7%	40.1%	+36.4%
SQLNet	12.4%	52.1%	+39.7%
IRNet	54.7%	69.2%	+14.5%

分析：

- LLM 增强嵌入为所有基线模型都带来了巨大的性能提升，特别是对较弱的基础模型（Seq2Seq, SQLNet）提升超过 36 个百分点。
- 即使是较强的基线模型 IRNet，也获得了 14.5 个百分点的显著提升。
- 这表明 LLM 所提供的深层语义理解能力是 Text-to-SQL 任务中的一个通用且强大的特征增强器。

解码器层数的影响

论文研究了 Transformer 解码器层数对性能的影响，结果如表2.3所示。

表 2.3: 解码器层数对 SLENet 性能的影响

解码器层数	测试集准确率
2	70.1%
4	72.4%
6	**74.1%**
8	73.2%
10	71.8%

分析：

- 当层数从 2 层增加到 6 层时，性能随之提升，在 6 层时达到最优的 74.1%。
- 当层数继续增加至 8 层和 10 层时，性能开始下降，表明模型可能出现了过拟合。
- 这说明 6 层 Transformer 结构为 SLENet 在 Spider 数据集上提供了模型容量与泛化能力的最佳平衡点。

2.4　总结

论文《Leveraging Large Language Model for Enhanced Text-to-SQL Parsing》所提出的 SLENet 模型，通过巧妙地融合大语言模型的语义理解能力与语法约束解码机制，在 Text-to-SQL 领域取得了 state-of-the-art 的性能。

1. **提出了一种新颖的 LLM 与约束解码融合框架：**SLENet 不是简单地用 LLM 进行端到端生成，而是将其作为强大的语义前端，后接一个保证结构正确性的解码器，这种分工协作的模式有效结合了两种技术的优势。
2. **引入了基于搜索的提示优化策略：**通过从外部数据集中动态检索相关示例，显著增强了大语言模型对当前查询上下文和任务要求的理解，提升了输入的信息密度和质量。
3. **设计了高效的语法约束解码器：**该解码器通过 SQL 语法掩码机制，确保生成的每一步都符合语法规则，从根本上避免了语法错误，提高了输出 SQL 的可靠性。
4. **实现了显著的性能提升：**在权威的 Spider 基准上，SLENet 显著超越了包括 PICARD、SmBoP 在内的所有现有先进方法，验证了该框架的有效性。
5. **提供了深入的实证分析：**通过系统的消融实验，明确验证了 LLM 增强嵌入、解码器深度等关键组件对性能的贡献，为后续研究提供了宝贵的见解。

SLENet 的工作表明，在利用大语言模型处理如代码生成、语义解析等具有严格结构约束的任务时，**将 LLM 的开放域生成能力与任务特定的结构化约束相结合**，是一条非常有效的技术路径。未来工作可集中于将类似框架应用于更复杂的 SQL 场景（如多轮对话式查询）、探索更高效的提示工程方法，以及研究如何降低对大规模 LLM 的依赖以提升实用性。

第三章　基础知识补充

3.1　SQL 概述

SQL（Structured Query Language，结构化查询语言）是一种用于管理和操作关系型数据库的标准化语言。它最初由 IBM 在 1970 年代开发，现已成为访问和操作数据库系统的通用语言。SQL 允许用户执行数据查询、插入、更新、删除以及数据库模式定义和访问控制等操作。

- **声明式语言**：用户只需指定“做什么”，而不必描述“如何做”
- **标准化**：ANSI/ISO 制定了 SQL 标准（如 SQL-92、SQL:1999、SQL:2016 等）
- **多功能性**：集数据定义、数据操纵、数据控制功能于一体
- **跨平台性**：大多数主流数据库系统（如 MySQL、Oracle、SQL Server、PostgreSQL 等）都支持 SQL

SQL 支持多种数据类型，包括：

- **数值类型**：INT, DECIMAL, FLOAT 等
- **字符串类型**：CHAR, VARCHAR, TEXT 等
- **日期时间类型**：DATE, TIME, DATETIME 等
- **布尔类型**：BOOLEAN

3.2　SQL 相关基准测试

简单来说，SQL 相关基准测试的核心作用是为 Text-to-SQL（将自然语言转换为 SQL 查询）这项技术提供一个统一、客观的“竞赛场”和“测量尺”。

3.2.1　ATIS 基准测试

ATIS（Air Travel Information System）是最早的 SQL 相关基准测试之一，起源于 1990 年代的 DARPA 项目。它专注于航空旅行领域，涉及航班查询、预订、票价等查询任务。虽然规模相对较小，但它在早期自然语言接口到数据库（NLIDB）研究中发挥了重要作用。特点如下：

- **领域特定**：专注于航空旅行领域
- **规模较小**：包含约 5,000 个自然语言查询及其对应的 SQL 语句

- **简单查询**：大多数查询是单表查询或简单的多表连接
- **历史意义**：为后续基准测试的开发奠定了基础

示例：

Listing 3.1: ATIS 查询示例

```
自然语言："显示从波士顿到西雅图的直飞航班"
SQL: SELECT flight_number FROM flights
     WHERE origin='Boston' AND destination='Seattle'
     AND stops=0
```

应用与局限：ATIS 主要应用于早期 NLIDB 系统的评估，但其领域限制和简单查询结构使其难以评估现代复杂 SQL 生成系统。不过，它仍然是评估模型在特定领域性能的有用基准。

3.2.2 GeoQuery 基准测试

GeoQuery 基准测试专注于地理信息领域，包含关于美国各州、城市、河流、山脉等地理实体的查询。它最初由威斯康星大学麦迪逊分校开发，是评估 NLIDB 系统在特定领域性能的常用基准。关键特点:

- **地理领域**：专注于美国地理信息
- **中等复杂度**：包含聚合、嵌套查询等较复杂结构
- **多语言支持**：有英语、西班牙语、土耳其语等多个语言版本
- **标准化数据集**：包含约 880 个查询-语句对

示例:

Listing 3.2: GeoQuery 查询示例

```
自然语言："德克萨斯州的首府是什么？"
SQL: SELECT city.name FROM city, state
     WHERE city.capital='yes' AND state.name='Texas'
     AND city.state_id=state.state_id
```

应用与局限:GeoQuery 广泛应用于 NLIDB 系统的评估，特别是在研究跨语言 SQL 生成方面。其主要局限是领域特定性和相对较小的规模。

3.2.3 WikiSQL 基准测试

WikiSQL 是由斯坦福大学于 2017 年发布的大规模基准测试。它专注于基于单个表格的 SQL 查询生成，是首个大规模、跨领域、众包构建的 SQL 基准测试。关键特点:

- **大规模**：包含超过 8 万个查询-表格-语句三元组
- **跨领域**：覆盖维基百科的各种主题
- **单表查询**：所有查询都基于单个表格

- **众包构建**：通过亚马逊土耳其机器人平台（Amazon Mechanical Turk）构建
- **结构简单**：不支持 JOIN 操作，但包含 WHERE、SELECT、AGGREGATION 等

示例:

Listing 3.3: WikiSQL 查询示例

```
表格：电影（电影ID，片名，年份，导演，票房）
自然语言："2010年哪部电影的票房最高？"
SQL: SELECT 片名 FROM 电影 WHERE 年份=2010
     ORDER BY 票房 DESC LIMIT 1
```

应用与局限:WikiSQL 因其大规模和跨领域特性，成为训练和评估神经 SQL 生成模型的流行基准。其局限性在于只支持单表查询，无法评估复杂的多表连接能力。

3.2.4 Spider 基准测试

Spider 是耶鲁大学于 2018 年发布的大规模跨领域复杂文本到 SQL 基准测试，被认为是当前最全面、最具挑战性的 SQL 基准测试之一。它包含 10,181 个查询和 5,693 个唯一的 SQL 语句，覆盖 200 多个不同领域的数据库。关键特点:

- **高复杂度**：包含各种复杂 SQL 结构，如嵌套查询、集合操作、多表连接等
- **跨领域**：覆盖 200 多个不同领域的数据库
- **跨数据库**：评估模型在未见数据库上的泛化能力
- **多表模式**：数据库包含多个相关表格
- **多难度级别**：查询分为简单、中等、困难、极难四个级别

复杂 SQL 示例:

Listing 3.4: Spider 复杂查询示例

```
自然语言："找出每个国家销售额排名前三的产品"
SQL: SELECT t1.country, t1.product, t1.sales
     FROM sales t1
     WHERE (SELECT COUNT(*)
            FROM sales t2
            WHERE t2.country = t1.country
            AND t2.sales >= t1.sales) <= 3
     ORDER BY t1.country, t1.sales DESC
```

评估方式:Spider 采用精确匹配（Exact Match）和组件匹配（Component Matching）两种评估方式。Spider 已成为评估复杂文本到 SQL 模型的黄金标准，其复杂性和跨数据库设置能有效评估模型的泛化能力。主要局限是构建和扩展成本高昂。

- **精确匹配**：生成的 SQL 在语法和语义上与参考答案完全一致
- **组件匹配**：比较 SELECT、WHERE、GROUP BY 等子句的匹配程度

3.2.5　发展历程与趋势

表 3.1: 四个 SQL 基准测试比较

特征	ATIS	GeoQuery	WikiSQL	Spider
发布时间	1990s	1990s	2017	2018
规模	小	小	大	大
查询数量	~5,000	~880	87,726	10,181
领域	航空旅行	地理	跨领域	跨领域
SQL 复杂度	简单	中等	简单	复杂
多表支持	有限	是	否	是
跨数据库评估	否	否	否	是
主要用途	早期 NLIDB	领域特定 NLIDB	神经 SQL 生成	复杂 SQL 生成

从 ATIS 到 Spider，SQL 基准测试的发展呈现以下趋势：

1. **规模扩大**：从数千个示例发展到数万个示例
2. **复杂度增加**：从简单单表查询到复杂多表操作
3. **领域扩展**：从特定领域到跨领域评估
4. **泛化能力**：从单一数据库到跨数据库评估
5. **现实性增强**：从人工构造到基于真实数据

这些基准测试对 SQL 生成研究产生了深远影响：

- 推动了序列到序列模型、图神经网络、预训练语言模型等在 SQL 生成任务中的应用
- 促进了跨领域、跨数据库的泛化能力研究
- 启发了新的评估指标和方法的发展
- 为工业界提供了评估自然语言数据库接口的标准化框架

3.3　自然语言与 SQL 之间的鸿沟

从自然语言输入合成 SQL 查询以查询关系数据库的 Text-to-SQL 转换任务，因其使数据库交互更易于访问和直观的潜力而受到相当多的关注。诸多公共基准测试表明，现有的 SQL 合成方法可以达到很高的精确匹配准确率。然而，随着现实世界数据集的复杂性日益增加，跨领域模式、复杂的 SQL 结构和未见过的数据库将对 Text-to-SQL 转换任务构成重大挑战，这些表明需要我们有技术上的推进，以有效弥合自然语言和结构化查询表示之间的差距。

- **更大规模**：需要更多样化、更大规模的数据
- **更复杂查询**：支持更复杂的业务逻辑和数据分析查询
- **交互式场景**：支持多轮对话式的查询生成
- **多模态支持**：结合文本、表格、图表等多模态信息

- **隐私保护**：在保护敏感数据的前提下构建基准测试

复杂 SQL 查询（如嵌套查询、多表连接、集合操作等）对模型提出了更高要求：

- **模式对齐**：将自然语言中的实体与数据库模式正确关联
- **关系建模**：理解表格之间的外键关系和连接条件
- **结构生成**：生成符合 SQL 语法的层次结构
- **上下文依赖**：处理跨句子和跨子句的依赖关系

自然语言到 SQL 的转换（Text-to-SQL）是自然语言处理与数据库系统的交叉领域，其核心挑战在于弥合自然语言的灵活性与 SQL 的结构性之间的鸿沟。传统方法在处理复杂查询、多表连接和嵌套结构时面临显著困难。为解决这些问题，IRNet 和 RAT-SQL 等先进技术引入了中间表示和关系感知的 Transformer 架构，显著提升了模型在复杂场景下的表现。

3.3.1　IRNet：基于中间表示的 SQL 生成

IRNet 是耶鲁大学团队在 2019 年提出的 SQL 生成模型，它在 Spider 基准测试上取得了当时的最佳性能。IRNet 的核心思想是引入**中间表示（Intermediate Representation, IR）**作为自然语言与 SQL 之间的桥梁，从而降低直接生成 SQL 的难度。IRNet 定义了一种**模式链接图（Schema Linking Graph）**的中间表示，将自然语言与数据库模式关联起来。IRNet 的中间表示包含两种链接关系：

1. **列-值对齐**：将自然语言中的值映射到数据库中的列
2. **列-列对齐**：识别需要连接的列对

IRNet 采用两阶段生成架构。阶段一：中间表示生成 阶段二：SQL 生成基于中间表示，IRNet 使用基于序列到序列的模型生成 SQL：

$$P(\text{SQL}|\text{IR}, Q) = \prod_{i=1}^{n} P(\text{token}_i|\text{IR}, Q, \text{token}_{<i})$$

关键技术组件

- **模式链接器**：基于 BERT 的编码器，计算自然语言与数据库模式的相似度
- **图神经网络**：在模式链接图上传播信息，增强列表示
- **解码器**：基于 LSTM 的解码器，生成 SQL 的抽象语法树
- **语法约束**：在解码过程中加入 SQL 语法约束，确保生成有效的 SQL

优势：

- 通过中间表示简化了生成任务
- 显式建模了模式链接关系
- 在 Spider 基准测试上取得了显著提升

局限：

- 中间表示的设计对任务性能有重要影响
- 两阶段架构可能导致误差传播
- 对复杂嵌套查询的处理仍有改进空间

Algorithm 1 IRNet 中间表示生成

输入：自然语言 Q，数据库模式 $S = \{(t_i, c_{i1}, c_{i2}, \ldots)\}$
输出：中间表示 IR
for 每个数据库列 c **do**
    计算 Q 与 c 的语义相似度
    if 相似度超过阈值 **then**
        标记 c 为相关列
    end if
end for
for 自然语言中的每个值 v **do**
    查找可能的值列匹配
    构建列-值对齐边
end for
for 每对相关列 (c_i, c_j) **do**
    基于外键关系和语义计算连接权重
    构建列-列对齐边
end for
返回完整的模式链接图 IR

3.3.2 RAT-SQL：关系感知 Transformer

RAT-SQL 是微软研究院在 2020 年提出的模型，它基于 Transformer 架构，通过**关系感知自注意力（Relation-Aware Self-Attention）**机制显式建模数据库模式中的结构关系。RAT-SQL 直接在统一的关系图上编码自然语言和数据库模式，无需显式的中间表示。RAT-SQL 的核心创新是扩展了标准的自注意力机制，使其能够感知输入元素之间的关系。关系感知自注意力的计算公式为：

$$\text{Attention}(Q, K, V, R) = \text{softmax}\left(\frac{QK^T + \phi(R)}{\sqrt{d_k}}\right)V$$

其中 $\phi(R)$ 是关系编码函数，将关系信息融入注意力权重计算。编码过程：

1. **输入表示**：将自然语言和数据库模式合并为单个序列

$$X = [[\text{CLS}], Q_1, \ldots, Q_n, [\text{SEP}], t_1, c_{11}, \ldots, c_{1m}, \ldots, t_k, c_{k1}, \ldots, c_{kp}]$$

2. **关系编码**：为每对元素计算关系向量 r_{ij}
3. **关系感知编码**：应用多层级联的关系感知 Transformer 层
4. **解码生成**：基于编码表示自回归生成 SQL

关键技术特点：

- **端到端学习**：无需中间表示，直接学习从自然语言到 SQL 的映射
- **显式关系建模**：通过关系编码显式捕获数据库结构

- **上下文感知**：利用预训练语言模型（如 BERT）的上下文表示
- **可扩展性**：可轻松扩展到更大的数据库模式

优势：

- 端到端架构减少了误差传播
- 显式的关系建模提高了模式链接的准确性
- 在复杂查询上表现优异，特别是在多表连接和嵌套查询上

局限：

- 计算复杂度较高，特别是对于大型数据库模式
- 需要大量的训练数据
- 对稀有关系的泛化能力有限

3.3.3　评价

IRNet 和 RAT-SQL 代表了 Text-to-SQL 领域的重要技术进步，分别通过中间表示和关系感知 Transformer 机制，有效弥合了自然语言与 SQL 之间的结构差异。

IRNet 通过引入模式链接图作为中间表示，将复杂的 SQL 生成任务分解为可管理的子任务，显著提升了模型在复杂查询上的表现。其两阶段架构为处理模式对齐和结构生成提供了清晰的框架，但在误差传播和端到端优化方面存在局限。

RAT-SQL 则通过关系感知自注意力机制，在统一的关系图中编码自然语言和数据库模式，实现了端到端的学习。其创新性的关系编码方法显式建模了数据库结构，在复杂查询特别是多表连接和嵌套查询上取得了显著优势。

两种技术各有侧重，IRNet 强调任务分解和中间表示的设计，RAT-SQL 注重端到端学习和关系建模。它们的成功不仅推动了 Text-to-SQL 技术的发展，也为更广泛的自然语言到结构化语言的转换任务提供了重要启示。随着大语言模型和多模态学习的发展，这些基础技术将继续演进，推动智能数据访问和自然语言数据查询的进一步发展。

3.4　SmBoP,PICARD,T5,Codex

神经模型在应对这些挑战的方面，取得了实质性进展。IRNet 和 RAT-SQL 等技术利用中间表示和关系感知 Transformer 来解决自然语言语法和 SQL 结构之间的差异，而 SmBoP 和 PICARD 等模型采用先进的解码策略来增强 SQL 生成并确保语法正确性。T5 和 Codex 等预训练语言模型通过利用迁移学习和少样本学习能力，进一步设定了新的性能基准，展示了它们在多样化数据集上的泛化能力。特别是在跨领域场景中，分层模式表示（如 HSRNet）和中间解析技术的进步改善了模式理解并简化了 SQL 生成。

3.4.1　解码策略的革新：确保语法正确的生成

传统序列到序列模型在生成 SQL 时，通常以自回归的方式逐个预测词元（token）。这种方式虽然灵活，但容易生成不符合 SQL 语法规范的序列，例如括号不匹配、缺少关键字或子句结构错误。为了解决这一问题，研究者提出了更具约束性的解码策略。

首先，让我们了解 SmBoP 模型。SmBoP 是“半马尔可夫波束搜索解析”的缩写，其核心创新在于将生成目标从线性的词元序列转变为更为结构化的“片段”。在标准的解码过程中，模型每一步预测一个词元。而 SmBoP 则允许模型在一步中预测一个完整的语法片段，例如一个完整的列名“user_id”，或者一个运算表达式“COUNT(*)”。这种半马尔可夫的性质使得解码过程能够更直接地遵循 SQL 的语法结构。模型在解码时，会同时考虑片段的语义内容及其在正在构建的抽象语法树中的位置。这种方法不仅加快了推理速度，更重要的是，它通过约束每一步的生成选项必须是语法上有效的片段，从而在解码过程中就大幅降低了产生语法错误 SQL 的可能性，从根源上提升了生成代码的结构正确性。

其次，PICARD 模型代表了另一种确保语法正确的思路，其名称为“通过自动回归解码进行解析的代码约束”。PICARD 并非一个独立的端到端模型，而是一个可以“套用”在现有序列到序列模型之上的解码时约束框架。它的工作原理是在标准的自回归解码过程的每一步，都引入一个代码语法约束检查器。具体而言，每当模型生成一个词元，PICARD 会根据当前已生成的部分序列，计算所有可能的后续词元集合，但这个集合会立即与一个预定义的 SQL 语法规则库进行交叉验证。语法规则库定义了在给定上下文中，哪些词元是合法的。例如，在“SELECT”关键字之后，合法的后续词元可能是列名、“*”或者“DISTINCT”等，而不可能是“FROM”。PICARD 会强制将解码器的概率分布中，不符合语法规则的词元概率设为零，从而引导波束搜索仅在语法正确的路径上进行。这种在解码时进行即时语法过滤的方法，能够非常有效地阻止语法错误的发生，显著提升了生成 SQL 的语法正确率，且对底层模型架构的侵入性较小。

3.4.2　预训练语言模型的革命：迁移与泛化能力

随着大规模预训练语言模型的兴起，文本到 SQL 任务迎来了新的范式转变。这些模型在海量通用文本或代码语料上进行了预训练，掌握了丰富的语言知识和一定的逻辑推理能力，通过迁移学习能够快速适应下游任务。

首先，T5 模型体现了一种统一的文本到文本转换思想。T5 将所有自然语言处理任务都重新定义为文本到文本的格式。对于文本到 SQL 任务，模型的输入是将自然语言问题和数据库模式信息（如表名、列名及其类型）按特定模板拼接而成的一段文本。模型的输出则直接是目标 SQL 查询语句的文本。这种设计的优势在于其简洁性与通用性。通过在大量文本语料（如 C4 数据集）上进行预训练，T5 学到了强大的语言理解和生成能力。当在文本到 SQL 的特定数据集（如 Spider）上进行微调时，它能够将预训练中获得的知识迁移过来，学习如何将一种特定格式的输入文本映射到 SQL 输出文本。T5 及其变体展示了预训练模型在理解任务指令和生成结构化输出方面的强大潜力，并在多个基准上取得了优异的性能。

其次，Codex 模型则将焦点从通用文本转移到了代码本身。Codex 是 OpenAI 在大量公开的 GitHub 代码库上微调 GPT-3 得到的模型，其核心能力是理解和生成编程代码。由于 SQL 本身也是一种领域特定语言，Codex 在代码数据上的预训练使其天然具备了理解 SQL 语法、常见模式和惯用法的能力。Codex 的应用开启了文本到 SQL 的“少样本”甚至“零样本”学习新范式。在少样本设定下，用户只需在提示中提供少量自然语言查询与对应 SQL 的示例，Codex 就能根据新的问题类比生成相应的 SQL。在零样本设定下，用户只需给出清晰的问题描述和数据库模式，Codex 也能凭借其对代码和语言的深刻理解生成可用的查询。这极大地降低了对特定领域标注数据的依赖，展示了大型预训练模型强大的泛化和类比推理能力。

3.4.3　跨领域理解的深化：模式表示与中间解析

在处理跨数据库、跨领域的文本到 SQL 任务时，一个核心挑战是如何让模型快速且准确地理解一个全新的、从未见过的数据库模式。以 Spider 基准为例，模型需要在测试时面对训练阶段完全未出现过的数据库及其表结构。这对模型的模式理解与泛化能力提出了极高要求。

为了解决这一挑战，研究者们致力于改进数据库模式的表示方法。分层模式表示网络是这一方向的典型代表。该方法的核心思想是构建一个层次化的图结构来表示数据库模式，而非简单的扁平化列表。在这个层次图中，数据库本身作为根节点，其下的各个表作为子节点，而每个表下的列则作为表节点的子节点。同时，图中还通过边来显式地标注表与表之间的外键关系、列与列之间的主外键引用关系，以及语义相似性关系。通过图神经网络在这一结构上进行信息传播和聚合，模型能够学习到每个表、列在全局模式上下文中的丰富表示。例如，一个“学生 ID”列，不仅知道自己属于“学生”表，还能通过图结构感知到它可能被“选课”表所引用。这种层次化和关系化的表示，极大地增强了模型对陌生数据库模式的理解和推理能力，使其能更准确地将自然语言中的提及（如“学生的名字”）链接到正确的模式元素上（“学生. 姓名”列）。

除了改进表示，另一个思路是引入中间解析技术来简化最终的 SQL 生成任务。这种方法不要求模型直接从自然语言一步生成复杂的 SQL，而是先将其转换为一种中间的、抽象的语义表示。这种中间表示比自然语言更结构化，但比具体的 SQL 语法更简洁、更接近语义核心。例如，中间表示可能明确标识出查询的意图、涉及的实体、过滤条件、聚合函数以及连接关系，但暂不决定具体的 SQL 关键字顺序或表别名。然后，在第二阶段，模型再根据这个清晰的中间表示，结合目标数据库的具体语法规则，生成最终的 SQL 语句。这种分而治之的策略，将复杂的映射问题分解为两个相对简单的子问题：语义解析和结构生成，从而降低了整体学习难度，并提升了模型在跨领域场景下的鲁棒性和准确性。

3.4.4　总结与展望

综上所述，文本到 SQL 技术的发展是一个多路径并进、不断融合的过程。SmBoP 和 PICARD 等模型从解码过程入手，通过引入语法约束来确保生成结果的结构正确性。T5 和 Codex 等大型预训练模型则从模型本身的能力出发，利用在海量数据上学到的知识和少样本

学习能力，实现了性能的飞跃和泛化能力的质变。面对跨领域这一核心挑战，以分层模式表示为代表的方法致力于提升模型对未知数据库的理解能力，而中间解析技术则试图通过任务分解来降低生成复杂度。这些技术进步共同推动着文本到 SQL 系统向更可靠、更通用、更实用的方向迈进。

尽管取得了这些技术上的进步，关键挑战仍然存在。当前方法仍然难以处理自然语言表达的多样性、多表查询的复杂性以及泛化到未见过的数据库模式的需求。这些限制凸显了持续创新的必要性，因为现有方法在完全弥合现实应用中的自然语言和 SQL 之间的语义差距方面常常存在不足。强调现实和复杂场景的 Spider 基准测试作为一个关键的评估指标，凸显了这些持续存在的挑战以及对鲁棒解决方案的需求。

第四章　基于人类认知过程增强大语言模型在 Text-to-SQL 翻译中的框架

4.1　引言

关系数据库存储大量数据，但用于从这些数据库检索数据的工具——结构化查询语言 (SQL) 传统上仅限于专家用户使用。将自然语言问题转换为 SQL 查询（称为 Text-to-SQL）多年来一直是研究重点。这对于创建更用户友好的数据库接口至关重要。

最近，卓越的大语言模型在问答和数学推理等各种任务中表现出色，也显著提升了 Text-to-SQL 任务的能力。直接指导 LLM（无论是开源还是专有）已成为有利的范式。这种方法避免了昂贵的微调成本，易于在 LLM 产品间过渡，并受益于其性能升级。当前该领域的研究旨在改进对自然语言问题的理解或提升生成 SQL 的质量。例如，任务分解用于增强 LLM 对简单子任务的注意力；基于检索的少样本示范用于激发 LLM 的上下文学习能力；任务对齐用于减轻 LLM 在 SQL 生成中的幻觉；多智能体协作用于共同提升 Text-to-SQL 质量。虽然这些策略提升了性能，但它们通常集中于特定的细微方面。

相比之下，人类以整体视角处理 Text-to-SQL，使用跨多个方面的推理来实现准确可靠的翻译。这种认知范式涉及在不同步骤间建立联系和过渡以确保精确性和可靠性。

人类应用过渡逻辑思维以集成方式将文本转换为 SQL 的三阶段认知过程：

- 准备阶段：识别基本概念，关注问题解决的关键部分，如相关数据库模式和语法关键词；
- 生成阶段：使用这些关键概念起草 SQL，通过分解分析其结构，注意详细的 SQL 子句和子查询；
- 校正阶段：审查起草的 SQL，考虑语法和与原始自然语言问题意图的一致性等方面，进行必要校正

这引出一个关键问题：LLM 能否从人类认知过程中受益以提高 Text-to-SQL 准确性？因此，本文提出 COGSQL 框架，旨在模仿人类认知过程以增强 LLM 在 Text-to-SQL 中的表现。分为三个主要模块：

- SQL 准备：识别相关数据库模式和语法关键词，类似于回忆关键概念；
- SQL 生成：使 LLM 能够像人类一样解释自然语言问题，并通过概念增强的思维链提示精确构建 SQL；
- SQL 校正：指导 LLM 反思生成的 SQL，校正语法错误并与自然语言问题意图对齐。

我们在五个大型具有挑战性的跨领域 Text-to-SQL 数据集上使用不同 LLM 进行实验，包括强大的专有模型和微调的开源模型。结果验证 COGSQL 有效增强了基于 LLM 的 Text-to-SQL 性能，并在各种 LLM 间具有良好的泛化能力。

4.2 相关工作

4.2.1 Text-to-SQL 进展

将自然语言问题转换为 SQL 长期以来一直是数据库和 NLP 研究的重点。早期方法依赖劳动密集的手工制定规则。神经网络的出现推广了编码器-解码器架构，其中编码器处理自然语言问题和数据库模式，解码器生成 SQL 查询。基于 Transformer 的架构和预训练语言模型进一步改进了 Text-to-SQL 性能。最近，LLM 展示了卓越能力，仅推理方法提供了卓越的可转移性。

4.2.2 SQL 准备

通过收集关键信息并执行自然语言问题-模式链接来增强 LLM 的生成过程。模式链接可以通过使用交叉编码器进行粗粒度链接，或通过示范和指令进行细粒度链接，提高翻译准确性。此外，示范选择通过选择自然语言问题-SQL 示例来优化 LLM 的上下文学习至关重要。

4.2.3 SQL 生成

SQL 生成涉及基于 SQL 准备中的关键信息进行复杂推理。当有效提示时，LLM 在这些任务中表现出色。思维链、分解和规划以及工具整合等技术极大地增强了 LLM 的生成能力。研究已将专门针对 Text-to-SQL 的思维链应用于实例。

4.2.4 SQL 校正

通过自一致性、自校正和通过执行结果验证等技术实现生成 SQL 的错误校正以确保可靠性。不适当的自校正可能降低准确性，自一致性由于重复的 LLM 推理可能在计算上昂贵。

4.3 预备知识

4.3.1 问题定义

设自然语言问题为 $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$，其中每个 q_i 是词元，对应的数据库模式为 $\mathcal{D} = \langle \mathcal{T}, \mathcal{C}, \mathcal{R} \rangle$，其中：

- $\mathcal{T} = \{t_1, ..., t_n\}$，每个 t_i 代表数据库中的一个表
- $\mathcal{C} = \{c_1^1, \ldots, c_1^{|t_1|}, c_2^1, \ldots, c_2^{|t_2|}, \cdots, c_n^1, \cdots, c_n^{|t_n|}\}$，每个 c_i^j 代表第 i 个表中的第 j 列

- $\mathcal{R} = \{(c_k^i, c_h^j) \in \mathcal{C}^2\}$，每对 (c_k^i, c_h^j) 表示列 c_k^i 和 c_h^j 之间的外键关系

Text-to-SQL 的目标是将 $\mathcal{Q}$ 转换为在 $\mathcal{D}$ 上可执行的 SQL 语句以检索所需结果。

4.3.2　语法关键词分类

表 4.1: 使用的语法关键词分类

类别级别	语法关键词
子句	SELECT, FROM, WHERE, ORDER BY, GROUP BY, HAVING
聚合	COUNT, SUM, AVG, MIN, MAX
字符串	SUBSTR
日期	STRFTIME
条件	CASE WHEN, IIF
空值相关	IS NULL, IS NOT NULL
比较	IN, BETWEEN, LIKE
算术	除法 ('/')
其他关键词	DISTINCT, CAST

4.4　COGSQL 方法

4.4.1　框架概述

图 2 说明：COGSQL 框架通过三个关键模块模仿人类认知过程增强基于 LLM 的 Text-to-SQL：(I) 关键概念回忆：执行 (1) 由粗到精的模式链接和 (2) 语法关键词预测，从给定自然语言问题和模式中识别过滤后的模式和语法关键词；(II) 概念增强的 CoT 提示：采用两阶段提示方法，包含 (3) 自然语言问题解释和 (4)SQL 组合，在最终确定答案前制定分步计划；(III) 基于一致性的校正：基于 (5) 自然语言问题一致性和 (6) 结果一致性评估 SQL 答案并进行必要调整。

COGSQL 包含三个主要阶段：

关键概念回忆

回忆关键概念启动了人类对复杂推理任务的认知过程。在制定 SQL 之前，人类通常关注相关数据库项和语法关键词。相应地，我们提出由粗到精的模式链接和语法关键词预测。

由粗到精的模式链接　现实世界数据库的模式通常包含大量表和列，由于长度限制可能使 LLM 不堪重负并阻碍 SQL 生成。模式链接旨在识别模式的子集。它应全面包含 LLM 制定 SQL 所需的所有必要数据库项，并简洁以避免可能误导 LLM 生成的额外表和列。

当前基于 LLM 的 Text-to-SQL 研究主要使用两种策略：一种直接提示 LLM 选择相关表和列，另一种将模式链接视为分类任务：

$$\mathcal{Y}^{sl} = f\left(\text{Enc}(\mathcal{Q}, \mathcal{T}, \mathcal{C}); \theta^{sl}\right) \tag{4.1}$$

其中编码器 Enc(·) 将输入序列映射到基于 Transformer 架构的嵌入中，分类器 $f(\cdot;\theta)$ 输出每个数据库项的概率。

COGSQL 提出结合两者优势的由粗到精模式链接方法：首先使用分类器进行粗粒度模式链接过滤不必要的表和列，然后使用保留的模式提示 LLM 并通过从 k_1 个表中选择子集（如 k_3 个）进行细粒度模式链接。

语法关键词预测　我们将语法关键词预测视为与公式 (1) 中粗粒度模式链接类似的分类任务：

$$\mathcal{Y}^{kw} = f(Enc(\mathcal{I}, \mathcal{Q}, \hat{\mathcal{T}}, \hat{\mathcal{C}}); \theta^{kw}) \tag{4.2}$$

语法关键词的不平衡分布可能阻碍分类器性能。为缓解此问题，我们提出以样本为中心的增强，利用 LLM 从原始训练样本合成高质量的自然语言问题-SQL 对。

概念增强的 CoT 提示

为改进 LLM 的 SQL 生成，我们解决两个关键问题：解释自然语言问题意图和组合 SQL 查询。我们将解决方案纳入 CoT 提示过程，模仿人类在正式回应前心理排练关键概念的方式。

自然语言问题解释　参考图 1 中的示例自然语言问题。看到此问题时，人类首先分析其内容以识别查询目标和返回数据。对于具有嵌套查询的复杂自然语言问题，决定是否需要此类查询也至关重要。

SQL 组合　回到人类认知过程：解释自然语言问题后，人类识别关键概念如何适应各种 SQL 子句，如识别 WHERE 子句的过滤条件。为复制此过程，我们专注于分解和生成 SQL。

基于一致性的校正

LLM 在生成中已知会出现错误和幻觉。具体来说，两个常见错误可能阻碍 LLM 的 Text-to-SQL 翻译：(1) 自然语言问题误解：LLM 可能误解自然语言问题的意图，导致语义不正确的 SQL 查询；(2) 语法错误：LLM 可能违反 SQL 语法规则，导致不可执行查询。

自然语言问题一致性　某些自然语言问题包含细微的语义细节。忽略这些可能导致 LLM 的误解。COGSQL 执行自然语言问题一致性校正来解决此问题。

结果一致性 SQL 语言具有复杂的语法规则，违反这些规则可能导致不可执行的 SQL 查询。COGSQL 执行结果一致性校正以确保生成的 SQL 查询在语法上正确。此校正包括两个方面：首先建立校正规则，每个生成的 SQL 查询进行语法检查；其次，基于规则的校正后的 SQL 查询在数据库中执行，不可执行的查询将在执行期间触发异常，我们提示 LLM 使用相应的异常校正这些不可执行的 SQL 查询。

4.5 实验与评估

4.5.1 实验设置

我们通过综合实验解决以下研究问题：

- RQ1. COGSQL 在不同 LLM 和数据集上的性能与现有方法相比如何？
- RQ2. 每个 COGSQL 组件对最终 Text-to-SQL 翻译准确性的贡献是什么？
- RQ3. COGSQL 如何减轻 LLM 在 Text-to-SQL 翻译中容易犯的错误？

数据集

我们使用公认的 Text-to-SQL 基准评估 COGSQL：(1) Spider；(2) Spider 的变体：Spider-DK, Spider-Realistic, Spider-Syn；(3) BIRD。

基线方法

我们将 COGSQL 与各种最先进方法比较，包括 RESDSQL, CODES, DIN-SQL, MAC-SQL, TA-SQL, DAIL-SQL, DEA-SQL 和 SUPER-SQL。

评估指标

遵循先前研究，我们使用执行准确率和有效效率分数作为性能指标。

实现细节

我们对以样本为中心的增强使用温度 0.7，COGSQL 其他模块使用温度 0。Spider 和 BIRD 训练集中的每个样本增强一次，得到来自自然语言问题重写的 1,390 个样本和来自关键词添加的 2,665 个样本。使用两样本提示进行增强、概念增强 CoT 和自然语言问题一致性校正模块。

4.5.2 总体性能 (RQ1)

COGSQL 有效增强了 GPT4 基线在两个基准上的性能。即使使用 GPT-4o-mini 等适中模型，COGSQL 仍保持竞争力，超过使用 GPT4 的强基线如 DIN-SQL 和 DAIL-SQL。当配备更强的 GPT-4o 和 GPT4 时，COGSQL 优于广泛微调的基线 RESDSQL 和 CODES。

4.5.3 消融研究 (RQ2)

表 4.2: COGSQL 模块消融研究（执行准确率%）

方法	Spider	BIRD
COGSQL+GPT-4o-mini	84.20	56.26
- 无由粗到精模式链接	82.40	55.74
- 无语法关键词预测	83.50	55.02
- 无概念增强 CoT 提示	83.50	52.41
- 无自然语言问题一致性	83.70	55.93
- 无结果一致性	83.60	53.85

每个模块都对整体性能有贡献，任何移除都会导致下降。在 Spider 上，COGSQL 表现出鲁棒性，即使移除单个模块也能保持稳定性能，除了由粗到精模式链接模块。在更复杂的 BIRD 上，概念增强 CoT 提示和结果一致性模块至关重要。

4.5.4 细粒度分析 (RQ3)

表 4.3: GPT-4o-mini 基线和 COGSQL 增强的 GPT-4o-mini 的错误分析

错误类别	GPT-4o-mini	GPT-4o-mini+COGSQL
模式误用	123	75(↓48)
关键词误用	46	35(↓11)
嵌套查询误用	47	39(↓8)
自然语言问题误解	35	22(↓13)
语法错误	11	2(↓9)
总计	262	173(↓89)

模式误用是最常见的错误，表明在复杂模式和自然语言问题中准确识别必要表和列的困难。COGSQL 有效减少了所有类别的错误，证明了其遵循认知过程逐步解决 Text-to-SQL 的能力。

4.6 结论

大语言模型在各种自然语言处理任务中显著提升了性能，包括 Text-to-SQL。当前基于 LLM 的 Text-to-SQL 方案主要集中于改进自然语言问题的理解或提升生成 SQL 的质量。虽然这些策略有效，但它们通常只针对特定的细微方面。

相比之下，人类采用整体视角处理 Text-to-SQL，在多个步骤中应用过渡逻辑推理来获得最终答案。我们相信 LLM 可以利用人类认知过程在 Text-to-SQL 中实现更高准确性。虽然 LLM 将 Text-to-SQL 性能提升到新水平，但如 BIRD 等复杂基准继续挑战即使最先进的 LLM。本研究中，我们提出 COGSQL，一个旨在模仿人类认知过程以增强 LLM 推理能力的新框架。COGSQL 利用由粗到精的模式链接和语法关键词预测有效回忆关键概念，随后通过概念增强 CoT 提示进行精确 SQL 生成。基于自然语言问题和结果视角的一致性校正确保对最终输出的鲁棒可靠调整。跨多样化数据集和 LLM 的广泛实验验证了 COGSQL 的优越性，并确认了模仿人类认知的好处。

本文提出 COGSQL 框架，该框架包含一套定制模型和策略，旨在复制人类认知过程以增强基于 LLM 的 Text-to-SQL。COGSQL 包含三个关键模块：(1) SQL 准备：采用由粗到精的模式链接和语法关键词预测，类似于人类回忆和对齐关键概念以更好地理解；(2) SQL 生成：引入概念增强的思维链提示，增强 LLM 的自然语言问题解释和 SQL 组合能力，类似于人类起草 SQL 查询；(3) SQL 校正：开发自然语言问题一致性和结果一致性技术来校正各种错误，模仿人类评估和精炼推理的过程。我们在多样化基准和 LLM 上进行了广泛实验，结果和分析验证了 COGSQL 的有效性和泛化能力。