2025-2026 秋季学期 Python 语言程序设计课程报告

KAN for Weibo

从数据流看端到端流水线的抽象设计

***

2025-12-7

目录

一、引言

虚假新闻（fake news）的泛滥正在成为全球范围内最严峻的信息安全挑战之一。社交媒体的极高传播效率使得未经验证的内容能在极短时间内进入公众视野，进而影响舆论、操纵情绪，甚至引发社会动荡。已有研究指出，假新闻往往通过情绪化叙事、细节伪造或“半真半假”的事实组合来误导读者，其危害性不仅体现在信息层面，也体现在对社会信任体系的侵蚀上。因而，如何构建可靠、可扩展、可自动化部署的虚假新闻检测系统，已成为自然语言处理（NLP）与社会计算领域的重要研究问题。

在此背景下，Dun et al. 提出的 **Knowledge-aware Attention Network (KAN)** 模型提供了一种具有代表性的解决思路：不仅利用新闻文本本身的语义特征，还显式引入知识图谱中的实体（entities）及其一跳邻居（entity contexts），以减少实体歧义、补充外部世界知识，并通过 N-E 与 N-E2C 注意力机制来度量不同知识单元对判别任务的重要性。KAN 的设计直观而有效，展示了将知识图谱融入新闻判别过程的潜力，在多个真实数据集上取得了显著优于传统内容模型的结果。

尽管 KAN 证明了知识增强方法在虚假新闻检测中的有效性，但其原始设计仍存在若干与实际应用场景相关的限制，尤其是在中文社交媒体语境下。首先，论文中的整个体系是在英文环境下开发与评测的，其分词策略、实体提取方式以及知识图谱对齐机制都假设输入文本具备清晰的词界与较高的书面语稳定性。而在中文场景中，微博等平台的文本往往呈现口语化、碎片化与弱结构化特征，导致**实体发现与链接难度显著提升**，从而影响后续知识融合模块的有效性。

其次，原论文在模型描述中对于若干关键工程细节采取了抽象化处理。尽管 KAN 在形式上将实体上下文定义为知识图谱中的“一跳邻居”，并采用平均池化的方式获得其向量表示，但论文并未给出一系列关键的工程实现细节，例如：如何控制上下文规模、如何处理邻域稀疏或为空的实体、是否以及如何对邻居集合进行截断、以及在使用 Transformer 进行编码时应采用何种输入顺序等。

这些缺失的细节使得实体上下文模块的实现具有显著的歧义性，无法在诸如 Wikidata 这样规模大、结构复杂且邻域高度不均衡的真实知识图谱上被直接复现。最后，现有的 KAN 原始实现更多关注模型效果验证，而非工业级开发实践。在模块解耦、数据流设计、可复现性、可扩展性以及与第三方工具协同方面仍存在显著差距，使得直接将其应用于大规模中文数据环境并不现实。

基于上述局限，我们的工作从工程角度对 KAN 进行了系统化重构，重点贡献包括以下三点：

1. **中文本地化适配（Chinese Localization）**

   我们重新设计了分词、文本清洗、实体候选生成与链接策略，使 KAN 能够在中文社交媒体环境中稳定运行；并构建了适用于中文语料的实体词表、分词器路由、知识图谱访问策略以及缓存机制，从根本上解决了原论文方法在中文场景下的不可直接迁移性。

2. **高可维护、可扩展的工业架构（Industrial-grade Architecture）**

   通过模块化的数据流水线（preprocess → linking → KG 查询 → encoding → fusion → inference）、统一的配置系统、可复现的训练流程、语义一致的编码接口以及清晰的 API 边界，我们为 KAN 建立了一个可长期维护、易扩展、可独立部署的工程体系，使其具备成为真实应用组件的能力。

3. **补全论文中缺失的关键工程细节（Filling Implementation Gaps）**

   对原论文在实体上下文构建、上下文池化、降噪与截断策略、邻域一致性等方面的抽象描述进行具体化实现；同时在知识编码器、注意力融合、缓存策略等环节加入稳定化与一致性处理，使 KAN 的知识融合过程具有明确可复现的操作语义。

这些改进的目标并非提出新的模型，而是让 **KAN 以现代深度学习工程标准重生**：稳定、透明、可解释、可复现，并且适用于真实世界的虚假新闻检测任务。

二、背景

1. 从文本模型到知识增强模型的演进

虚假新闻检测（Fake News Detection）过去十年呈现出清晰的技术演进路线，从早期基于人工特征的方法，逐渐发展到深度神经网络，再到近年来的多模态融合与知识增强（KG-enhanced）模型。

1.1 纯文本模型（Text-based Models）

传统方法依赖词频、句法、情感等人工特征，后来逐步被 RCNN、GRU、Transformer 等深度模型替代——它们能捕获新闻内部的句法和语义结构，但**缺乏对外部世界知识的建模能力**。

1.2 多模态模型（Multimodal Models）

部分研究引入图像、用户特征或社交传播图（Wang et al., 2018b; Liu & Wu, 2018）。这些方法具有更丰富的输入，但其性能依然受限于：

- 新闻文本极为压缩，实体信息高度密集；
- 模型无法理解实体之间的逻辑关系。

1.3 知识增强模型（KG-enhanced Models）

知识图谱（Knowledge Graph, KG）为模型引入显式实体与关系结构，能有效弥补文本语义的缺失。B-TransE、KCNN 到 KAN 等模型证明了**实体消歧（entity disambiguation）与知识补充（knowledge completion）能显著提升检测性能**。

其中，**Knowledge-aware Attention Network（KAN）** 是代表性工作。KAN 通过两级注意力（N-E 与 N-E2C）衡量新闻、实体与实体上下文的相关性，从而融合知识层表示（q, r）与语义层表示（p）。

然而——KAN 的原始设计主要针对英文微博与英文 Wikidata 场景，其 pipeline 在中文环境下存在天然断层，这正是我们项目要接住的“落差”。

2. 为什么中文场景完全不同？

中文 NLP 与英文最大的结构性差异在于**中文缺乏显式词界（no explicit word boundary）**。这会导致两个直接后果：

2.1 中文分词本身就是一个难题

- “北京大学生物系学生会”这种结构天然含糊；
- 不同分词器结果分歧巨大；
- 分词错误将级联影响实体识别与实体链接。

因此，在英文中相对容易的“mention detection”到中文语境下会转变为**不稳定且高噪声的输入流**。

2.2 中文实体链接（EL）比英文难度更高

即便找到了候选 mention，EL 仍然受到以下限制：

- 同形词（多实体共享同一中文名称）；
- 中文别名极多（艺名、简称、历史名称等）；
- 社交媒体文本常出现非标准表达；
- 对应的 Wikidata 项中文 label 缺失或不规范。

基于上述原因，原论文假设的 “实体提取 → 精准消歧 → KG 查询” 在中文领域并不能直接套用。

3. 实体链接与中文知识图谱：资源不一致与 API 工程成本

更进一步，中文场景的 KG 使用还面临 **Wikidata 工程层面的现实限制**：

3.1 中文 Wikidata label/alias 不全面

导致链接准确率降低。

3.2 API 查询成本高、并发限制严苛

原论文默认每个实体做一次 one-hop 查询（ec(e) 获取邻域集合）。
但在真实工程里：

- 一条新闻动辄数十个实体（PHEME 数据集平均 20～55 个实体）
- 每个实体又可能拥有数百邻居
- SPARQL endpoint 有严格的限流

这一现实问题导致我们**必须重写 KG client**，实现：

- 本地缓存；
- 失败重试与降级；
- 受控并发；
- 限制邻域规模；
- 高吞吐的数据预取架构。

这些工程需求或许超出了原论文的描述范围，但是面对现实挑战的灵活权衡。

4. KG-enhanced 工程挑战：原论文 KAN 未充分讨论的部分

虽然知识图谱增强模型在概念上具有较强的可解释性，但在实际工程化落地时会遇到显著的结构性障碍。以 KAN 为代表的模型虽然提出了知识注意力机制，但其论文主要关注方法层面的结构创新，而未对工程执行细节做深入展开（如 entity context 构造、KG 查询开销等）。我们将这些未充分讨论但对系统可运行性至关重要的问题总结为三类。

4.1 大规模邻域

原论文在知识抽取部分仅给出了实体上下文 ec(e) 的定义，并说明将其邻居实体的向量做平均池化以得到上下文表示，但未进一步讨论邻域的规模控制、采样策略与工程可行性问题。

然而在真实的 Wikidata 中，一个实体往往拥有数百到数千一跳邻居。例如政治人物、热门机构等中心节点，其 degree 极高。若不加限制，会导致：

- **邻域爆炸（context explosion）**：每个实体带来 $O(10^3)$ 邻居。
- **上下文长度不受控**，Transformer 编码器输入被迫倍增。
- **内存与 batch size 崩溃**，无法在 GPU 上稳定训练。
- **延迟问题严重**：一条新闻若有 50 个实体，就可能触发 50 次高代价 KG 查询。

这一问题在中文场景下更为严重，因为中文实体链接的噪声更高，导致大量误链实体进入邻域，进一步放大数据规模与噪声污染。

4.2 噪声实体与错误对齐（Noisy or Wrong Entities）

实体链接错误会在 KG-enhanced pipeline 中产生级联放大效应。错误实体一旦进入 KAN，将导致：

- 获取到完全不相关的上下文
- Transformer 知识编码器被噪声干扰
- 注意力机制错误聚焦
- 最终分类器决策被污染

原论文假设 EL 工具能够提供高质量实体（在英文任务中这一假设较为可行），因此未提供任何形式的 **噪声抑制机制（noise filtering / sanity checking）**。

然而在中文场景，实体链接本身噪声较大且分布高度偏移。例如：

- 人名多义
- 组织名称缩略多变
- 缺乏标准化 mention

三、相关工作

1. TagMe 在中文场景下的局限性与替代

KAN 原论文使用的实体链接工具 TagMe 具有成熟的英文维基百科链接能力，但其对中文文本的支持极为有限。一方面，TagMe 的候选实体构建依赖于英文 Wikipedia 的标题、重定向与锚文本，这使其难以覆盖中文常见实体；另一方面，

其内部统计特征（如链接先验、共现概率）均基于英文语料构建，直接用于中文会导致召回率显著下降。

由于任务场景完全基于中文社交媒体文本，我们无法完全复用 KAN 原论文默认的 TagMe 链路，因此寻找替代设计。这形成了我们采用“分词后直接对接 Wikidata 查询”的工程动机：以分词结果为候选 mention，通过规则与实体字典查询获得可复现的中文实体链接能力。该方案虽较为传统，但保证了实体覆盖率与可控性。

2. 中文分词器的工程权衡：从 LTP 到 jieba

由于 TagMe 无法直接用于中文，实体链接部分对分词质量提出更高要求。主流的 LTP 分词器在社交媒体短文本中倾向于采用细粒度策略，将大量词语切割为单字。对于文本理解任务，这是一种可接受的建模决策；但对于实体链接而言，单字级 tokenization 会导致查询完全失效，例如“北京大学”被切成“北 / 京 / 大 / 学”后不再匹配任何知识库实体。

因此，我们在工程上采用了更传统的 jieba 分词器。jieba 在处理专名、机构名、人名时具有更稳定的词级粒度，显著提升了实体候选匹配的可用性。该决定是一种非算法创新，而是典型的工程层面的实用主义权衡：更适合中文实体链接链路，而不争取在语义建模上“最先进”。

3. 文本编码器选择：BERT 的优势与语义空间不对齐

BERT 是当前文本表示学习中的事实标准，与传统 Transformer Encoder 相比，其词表、预训练语料与深度堆叠结构都能显著提升文本语义的捕获能力。因此，本项目在文本流中引入 BERT 作为编码器以替代原论文的浅层 Transformer。这一改动显著增强了模型的文本表示能力，也使得本项目整体性能对小数据集更鲁棒。

然而，BERT 引入了新的问题：文本表示与实体 / 上下文表示处于不同语义空间。文本使用 BERT 子词词表，而实体与实体上下文仍依赖知识图谱中的离散 ID 及其 embedding。两类 embedding 并无天然可比性，因此 KAN 的两个注意力机制（N-E 与 N-E2C）在语义空间上缺乏严格对齐。出于工程完整性与兼容性，我们保留了实体与上下文的 Transformer Encoder，但用 RoPE（Rotary Positional Encoding）增强其序列建模性能，使其更接近 BERT 在序列结构上的能力。

综上，BERT 的引入是一项实践上的增强，但也引入了语义空间不一致的问题，而限于数据集大小这一实践问题，我们维持了实体流与上下文流的独立表示和随机初始化的 Transformer，仅对原始论文采用的位置编码进行了优化。

4. 优化器与调度器解决小数据集训练中的稳定性与过拟合

原论文 KAN 并未给出优化策略与调度器的细节，通常采用标准 Adam 优化器即可。但在本次小规模中文数据集上，模型深度提升以及 BERT 的引入显著增加了过拟合风险。

为缓解此问题，我们采用了两个工程方案：

1. AdamW 优化器用于防止权重衰减不当带来的训练不稳定性；
2. cosine learning rate scheduler（余弦退火调度器）用于柔和地降低学习率，改善小样本条件下的训练收敛行为。

这两项改动使模型在 3k 级中文数据上能够更平稳地收敛，并且降低了强模型在后期迅速过拟合的风险。

四、KAN 本地化实现

1. 设计思路与总体框架

本项目围绕一个统一的端到端映射来组织工程：给定一条中文新闻文本以及一个外部知识图谱（Wikidata），系统实现从原始文本到“假新闻概率”的整体映射

$$F\colon (RawText, KG) \to prob(fake)$$

其核心不在于单个网络层的细节，而在于**算子（operators）的组合顺序和数据流（data flow）**。因此，我们首先把 KAN 视为一个数据驱动的流水线系统，而不是孤立的神经网络模块。

1.1 数据流视角下的 KAN

从数据流角度看，KAN 的建模可以被分解为五个相互解耦但连续串联的阶段：

1. **原始数据阶段（Raw Data）**
   输入是一条微博新闻文本 $S = \{w_1, \dots, w_n\}$，以及一个包含实体和关系的外部知识图谱（Wikidata）。此时数据仍处于“自然语言 + 知识库”的原始形态。

2. **特征抽取与嵌入阶段（Feature Extraction & Embedding）**
   在这一阶段，原始数据沿着三条“并行通路”被转换为向量表示：

   - **News Stream（新闻流）**：对文本进行清洗、分词与编码，将词映射为词向量，并加入位置编码得到新闻输入表示 u。

   - **Entity Stream（实体流）**：通过实体链接将文本片段对齐到知识图谱中的实体，得到实体序列 E，再将其映射为实体嵌入 E'。

- **Context Stream（上下文流）**：对每个实体从知识图谱中提取一跳邻居，平均聚合得到实体上下文嵌入 EC'。
  这一阶段的目标，是把“文本 + 实体 + 实体上下文”统一压缩到数值向量空间，为后续编码器和注意力算子提供输入基础。

3. **编码阶段（Encoding via Transformer）**
   三路向量分别交由轻量级 Transformer Encoder 精炼：

   - $u \to p$：得到新闻语义表示 p；
   - $E' \to q'$：得到实体的中间表示 q'；
   - $EC' \to r'$：得到实体上下文的中间表示 r'。
     至此，我们获得了三种语义粒度不同、但维度对齐的表示，为后续的知识融合提供了统一接口。

4. **知识注意力融合阶段（Knowledge-aware Attention Fusion）**
   KAN 的关键思想是：并非所有实体与上下文对当前新闻同等重要，因此需要一个**显式、可解释的注意力机制**来建模知识的相对权重。我们延续原论文的两级注意力结构：

   - **N-E Attention（News → Entities）**：以新闻表示 p为 Query，以实体中间表示 q'为 Key/Value，通过多头注意力得到加权后的实体表示 q，反映“当前新闻更关注哪些实体”。
   - **N-E²C Attention（News → Entities & Contexts）**：仍以 p为 Query，以 q'为 Key、以上下文中间表示 r'为 Value，通过多头注意力得到加权后的上下文表示 r，反映“在重要实体周围的哪些知识邻居值得被强调”。
     这一阶段的本质，是用一个相对简单而通用的算子（Multi-Head

Attention），把**语义级表示**与**知识级表示**对齐并融合，而不是在网络结构上引入大量特化模块。

5. **决策阶段（Classification）**

   最后，将新闻表示 p、加权实体表示 q与加权上下文表示 r拼接成统一特征向量 $z = p \oplus q \oplus r$，输入到一个浅层全连接网络，输出“假新闻概率”作为最终预测结果。

从这个角度看，KAN 的核心是一条从“原始文本与知识图谱”到“概率预测”的**有向数据流图**：节点是算子，边是数据形态的演化。我们在工程上优先保证这条数据流的清晰与可追踪，然后再讨论具体的模型实现细节。

1.2 模块化与职责划分

在工程实现上，我们将上述数据流水线拆解为若干职责单一、边界清晰的模块，并以目录结构和 API 进行显式编码：

- **数据与知识图谱层（kan.data）**：负责从原始 CSV 文件与 Wikidata 读取数据，完成文本预处理、中文分词、实体链接、邻域查询与缓存等操作，将“现实世界输入”转换为结构化的 PreprocessedSample。

- **表示层（kan.repr）**：负责构建和管理词表/实体表、实现文本和实体的嵌入层以及批处理逻辑（batching），保证所有输入张量在维度和索引上的一致性。

- **模型层（kan.models）**：实现文本编码器（包括我们引入的 BERT 文本编码器）、知识编码器、注意力模块以及顶层 KAN 模型，将数据流阶段 2–5 的算子统一封装为一个端到端可调用的模型。

- **训练与评估层（kan.training）**：提供 Trainer 和 Evaluator，对上层暴露标准的训练/验证接口，将优化器、调度器、早停、日志记录等训练细节从模型本体中剥离。

- **工具与配置层（kan.utils）**：使用 dataclass + JSON 管理配置，统一处理随机种子、日志与指标计算，使不同实验配置可以在不破坏核心流水线的前提下灵活切换。

在此之上，我们构建了一个完全独立的命令行前端包 kan_cli，它只通过公开的库 API（如 KANConfig, KAN, NewsDataset, Preprocessor, KnowledgeGraphClient, Trainer 等）来驱动整个流水线，而不反向侵入 kan 内部。这种"库 / 前端"分离的设计，使得我们既可以在命令行环境下批量训练与预测，又可以在 Jupyter Notebook 或其他系统中将 KAN 作为普通 Python 库直接调用。

1.3 与原始 KAN 模型的关系

从算法层面，本项目继续遵循原始 KAN 的建模思想：利用外部知识图谱提供的实体及其上下文信息，通过 Transformer 与知识注意力机制同时建模语义层与知识层，从而提升假新闻检测性能。

然而，在实现层面，我们刻意将这一思想"工程化"与"本地化"：

- 用统一的数据流视角重构模型各子模块，保证不同编码器与注意力块之间的接口一致；

- 将实体提取、上下文获取、向量化和注意力融合明确拆分为流水线阶段，以便于在中文场景下替换分词器、重写实体链接与知识图谱客户端；

- 通过清晰的模块化结构和配置系统，使得后续引入 BERT 编码器、RoPE 位置编码或调度器等工程增强手段时，只需要调整"算子实现"而不破坏整体数据流结构。

2. kan.data

在 KAN 的整体架构中，kan.data 模块承担了**数据流的最底层职责**：它将原始、非结构化的新闻文本，逐步转化为可被 KAN 模型消费的三类关键输入：

1. **Token 序列 S（文本语义）**
2. **实体序列 E（知识实体）**
3. **实体上下文序列 EC（一跳邻居结构化知识）**

本节从工程角度呈现 kan.data 的内部流程、模块划分及其增强设计。

2.1 kan.data.datasets: 高可靠数据加载层

数据结构（NewsSample）抽象了一条新闻的数据单元：

- **id**：唯一编号
- **text**：原始文本
- **label**：训练集标签（测试集为空）

这是整个数据通道的“原材料”，它只承担数据容器（container）功能，不做任何预处理。

数据集加载（NewsDataset）类实现：

1. **CSV 解析**（支持任意字段名映射）
2. **批次生成器（batch_iter）**
3. **训练常用抽取接口（get_texts_and_labels）**

特别地，我们增强了以下工程特性：

- **无副作用（deterministic）加载逻辑**
- **可配置 shuffle 行为**
- **严格类型化的 dataclass** 提高可维护性
- **与上层 Preprocessor 完全解耦**

2.2 kan.data.preprocessing: 文本 × 知识的联合预处理层

这是本系统最重要的工程模块之一。

其目标是将 **NewsSample → PreprocessedSample**，即从非结构化转为结构化三元组：

$$(S, E, EC)$$

原论文仅概述流程（参见 Fig.3），但**没有说明工程可复现性与中文场景特殊性**。我们的实现做了大幅补全与优化。

文本清洗（Text Cleaning）包括：

- lowercase（可配置）
- URL 清理
- @用户提及清理
- 多空白折叠

这是为了减少实体链接噪声，提高分词质量。

我们提供了**多路分词器支持（Tokenizer Routing）**。虽然论文没有说明分词细节，但在中文场景中分词对实体链接精度影响极大。因此我们引入了可配置的三种 tokenizer：

tokenizer	特点	适用场景
jieba	工业级中文分词，词语粒度适中	实体链接最稳定（TagMe 不支持中文）
LTP	强语法模型，但倾向字级分词	不适合实体链接（我们因此弃用）
regex	英文/数字/CJK 单字切分	兜底策略

我们发现 LTP 会将中文切成单字，使得实体链接失败率极高。因此本项目改用 jieba，使 token 更接近 Wikidata 的 alias。

其中 **KG 联合处理：实体链接与上下文抽取**这一部分采用**异步 + 并发线程池**实现，是论文中最模糊但本项目最关键的工程改进。

实体链接（Entity Linking）

输入：分词结果 token list
输出：实体序列 E = {QID}

我们使用 Wikidata API（wbsearchentities）逐 token 查询并做如下优化：

- **长度≤1 的 token 自动过滤**（中文单字常常无有效语义，噪声极大）
- **正则过滤非字母/数字/CJK token**
- **表面形式缓存 surface_cache（JSON 持久化）**
- **搜索失败回写空列表避免重复请求**

这些策略保证性能与礼貌性，对公共 API 友好。

实体上下文抽取（Entity Context Retrieval）

输入：实体 E

输出：上下文 EC，每个实体的一跳邻居

利用 SPARQL 查询：

$$ec(e_i) = \{e \mid (e_i, rel, e) \text{ or } (e, rel, e_i)\}$$

我们的工程优化包括：

强化 1：多线程并发检索：知识图谱查询慢，我们利用 ThreadPoolExecutor 并发访问。

强化 2：邻居缓存（内存+磁盘）：减少对 Wikidata 的负载。

强化 3：稳定去重 + 索引映射：提升批处理效率——这也是原论文未提及却极其关键的部分：避免多次查询同一实体。

异步批处理（Batch-level KG Fusion）是 Preprocessing 模块的大杀器。源码中的：_run_batch_kg() 会对一个 batch 的样本同时发起成百上千个异步任务，使得 I/O 密集型的 Wikidata 查询速度得到数量级提升。这使得我们在真实数据集上**能在几十秒内完成预处理，否则会延迟数十分钟甚至超时**。

最终输出结构（PreprocessedSample）呈现如下：

```
@dataclass
class PreprocessedSample:
    id: int
    tokens: List[str]              # S
    entities: List[str]          # E
    entity_contexts: List[List[str]]   # EC
```

label: Optional[int]

这恰好对应论文的输入三元组 (S, E, EC)，并作为 KAN 中三路 transformer 编码器的输入。

2.3 kan.data.knowledge_graph：知识图谱客户端

该模块是实体链接与上下文查询的核心。

为**查询端点与礼貌访问**配置包括：

- SPARQL endpoint
- 超时时间
- User-Agent
- 最大邻居数
- 语言（中文场景设为 "zh"）

此处我们进行了针对中文任务的本地化适配。

API 设计（同步 + 异步统一封装）中该模块提供四类接口：

功能	同步 API	异步 API
实体链接	link_entities_from_tokens	alink_entities_from_tokens
邻居查询	get_neighbors	aget_neighbors
批量上下文	get_entity_contexts	aget_entity_contexts

所有异步接口底层通过线程池 offload 同步网络请求，避免 GIL 阻塞。

系统采用了**三层缓存**：

- **内存缓存（dict）**

服务单次训练过程，多线程安全。

- **磁盘缓存（JSON）**

跨训练 session，减少 API 负载。

- **原子写入策略**

通过锁保证一致性，避免并发时写坏 JSON 文件。

3. kan.repr

kan.repr 模块承担整个系统中关键的职责：**将原始符号序列（tokens、实体 ID、上下文实体 ID）转换为可进入 Transformer 与注意力机制的数值表示**。从数据流的角度来看，它是原始预处理样本进入 KAN 模型之前的“语义压缩点”，决定了后续编码器所能看到的信息结构。

原论文仅给出了高层抽象流程，例如：

- 文本编码采用 Transformer Encoder 生成 p；
- 实体编码生成 q'；
- 上下文池化得到 $ec'(e_i)$，并再通过 Transformer 生成 r'；
  原文明确指出上下文采用“一跳邻居平均池化” 。

然而，论文并未给出工程层面的细节，例如词表构造策略、PAD/UNK 对齐方式、上下文池化的异常处理、BERT 模式的兼容性等，这些均在本项目中通过 kan.repr 得到系统化补全与重构。

kan.repr 的设计目标包括：

1. **统一的数据表示接口**：文本、实体、上下文均转换为形状稳定、可直接送入模型的张量。

2. **与上游预处理（Preprocessor）和下游模型（KAN Encoder）保持 API 稳定性**。

3. **补全原论文未给出的工程细节**，特别是：

    - 实体上下文池化策略
    - PAD 掩码的一致性
    - BERT 与词表模式的共存
    - 无实体样本批次的零退化策略

接下来我们按照模块内部逻辑结构，分三部分展开：词表构建、嵌入层设计、批处理机制。

3.1 Vocab：符号空间的离散化建模

词表模块 vocab.py 的主要目标是将文本 token 与知识图谱实体 ID 映射为离散索引空间，从而提供给后续嵌入模块一个 **稳定且有限的整数域**。其设计原则包括：

1. **语料驱动（corpus-driven vocabulary）**：基于频率构建词表，提高训练稳定性与推理效率。

2. **PAD/UNK 显式建模**：所有下游模块依赖 PAD 掩码，因此 vocab 必须确保特殊符号的确定性。

3. **实体词表与文本词表隔离**：避免 ID 交叉污染，保持各自语义空间独立。

源码中 build_text_vocab 与 build_entity_vocab 的工厂函数从预处理样本构建词表 ，采用如下策略：

- 先插入特殊符号，固定顺序（PAD → UNK → BOS → EOS）
- 再按频率和字典序排序
- 控制最大词表大小与最小频率阈值

该过程保证了词表的确定性（determinism），避免因遍历顺序波动导致训练结果不稳定。

论文并未讨论词表构造，但由于实体 ID 本质上是离散的 Wikidata 编码，如 Q76、Q30，本项目的实体词表部分严格映射实体字符串本身。此策略与论文假设完全一致：实体嵌入空间由外部离散实体集合驱动。

3.2 TextEmbedding：文本表示的向量化

文本嵌入层 text_embedding.py 的功能是将离散 token ID 序列映射为具有位置编码的连续向量，从而作为 Transformer 输入。对齐 Transformer Encoder 的要求，它需提供：

- 稳定的嵌入维度 d_{model}
- 不可训练的正弦型位置编码
- 对 PAD token 的零向量处理

与原论文一样，本模块使用标准的 *Sinusoidal Positional Encoding*（不可训练）。为了保证输入长度超过 max_len 时的可读性，模块进行了：

- 序列长度检查
- 明确报错提示以提升可调试性

该设计延续 Transformer 经典结构，并保持参数可控。相比可训练位置编码，正弦编码在低数据量场景中更不易过拟合。此外，与后续 BERTEncoder 共存时，此模块仍能继续作为可选路径存在，不破坏配置兼容性。

3.3 EntityEmbedding：知识表示的向量化与上下文池化

实体嵌入层 entity_embedding.py 负责：

- 将实体 ID 映射为实体向量 e_i'
- 将上下文实体 ID 序列映射为 $ec'(e_i)$
- 对上下文实体执行池化（mean/max）

原论文仅说明“取邻居实体平均值作为上下文表示”，但实际数据存在多种异常情况，例如：

- 上下文实体数量不齐
- padding mask 缺失
- 某实体完全没有上下文（Wikidata incompleteness）

本项目对这些细节进行了补全过程，并提供可选 max pooling：

```
def _pool_context(...):
    # masked mean 或 masked max
```

特别重要的是：

当某实体没有任何上下文时，向量退化为全零，保持与 PAD 行为一致。这使得注意力模块能够自然忽略无上下文实体，无需额外判定语句。

出于节省参数规模与对齐语义空间的考虑，配置允许实体与上下文共享同一 embedding table，这是工程上的合理优化。

3.4 Batching：将样本批次化为张量

批处理模块 batching.py 是 kan.repr 的管线顶层，它将 PreprocessedSample 转换为 BatchEncoding，包括：

- 文本 ID 张量（Bert 或 Vocab 模式）
- 实体 ID 张量
- 上下文 ID 张量（$B \times L_e \times L_c$）
- 对齐的 padding mask

该模块的输出直接进入：

- TextEmbedding
- EntityEmbedding
- KAN 主模型

是整个数据表示流的核心节点。

(A) 文本路径双模设计：Vocab vs BERT

在 vocab 模式下：

- 直接使用 text_vocab.encode
  在 bert 模式下：
- 调用 HuggingFace tokenizer
- 支持“head” 与 “tail” 截断策略
- 明确 attention_mask 与 token_padding_mask 的对应关系

此设计使得原论文的词表模式与本项目的 BERT 试验路径能够并行共存，便于研究对比。

(B) 实体对齐与上下文对齐策略

源码中对齐逻辑如下 ：

- 若上下文数量少于实体数量 → 自动补空
- 若更多 → 截断
- 对齐后再根据 max_entities, max_context_per_entity 截断

这种机制保证所有批次的维度一致，不需要模型端处理异常。

(C) 无实体样本的零退化策略

当某个批次完全没有实体时，Batcher 会返回：

```
entity_ids=None
context_ids=None
```

并在 log 中提示：

Batcher.collate: batch has no entities at all.

下游模型随后采用“零向量退化”策略，使得 KAN 在无知识场景下退化为纯文本 Transformer，此策略保持与论文一致：知识是可选补充，而非严格依赖项。

4. kan.models

本节系统性解析本工程中 kan.models 的整体架构设计，包括文本编码器（Text Encoder）、知识编码器（Knowledge Encoder）、知识注意力模块（Knowledge-aware Attention）、以及最终分类器（Classifier Head）。这些模块共同构成了一个端到端的可微学习系统，将文本–实体–实体上下文三个不同模态的信息有效融合，实现对新闻真伪的精准预测。与原论文相比，本工程实现采用了更成熟、高质量、工业可维护的模块化结构，并针对中文数据场景、本地化需求和预训练模型生态进行了大幅优化。

为了确保工程可维护性与可扩展性，本项目的模型架构遵循以下设计原则：

1. **解耦（Decoupling）**：文本编码、知识编码、注意力与分类器均为独立模块，具备清晰输入输出边界。

2. **统一接口（Unified Interface）**：所有子模块均输出 shape=(B, L, D) 或 (B, D) 的张量，保证流水线可无缝拼接。

3. **单向依赖（Monotonic Dependency）**：上层模块不反向依赖下游，以保持模型在 CLI 与库模式下的可重用性。

4. **空间对齐（Space Alignment）**：所有向量表示在最终融合前均对齐到同一维度 D（等同于 BERT hidden size）。

5. **算子透明（Operator Transparency）**：任何一次维度变化、池化或注意力操作都在源码中明确定义，不隐藏副作用。

而 KAN 主模型由四类算子组成：

1. **文本语义编码（BERT Text Encoder）**：生成新闻表示 p。
2. **知识编码（Knowledge Encoder）**：生成实体编码 q′ 与上下文编码 r′。
3. **知识注意力（N-E 与 N-E²C Attention）**：生成加权实体表示 q 与上下文表示 r。
4. **分类头（Classifier Head）**：将 z = [p; q; r] 送入线性层得到 logits。

文本–实体–上下文三路信息经 Transformer 编码及注意力融合后在表示空间中汇合，实现新闻的知识增强语义建模。

4.1 底层算子实现

4.1.1 序列池化（pooling）

该模块提供两类基础池化算子：

1. **masked_mean_pool**：对有效 token 做均值池化，避免 padding 引入噪声。
2. **cls_pool**：取首位 token（如 BERT [CLS]）作为句子表示。

本工程中，由于新闻文本使用 BERT 进行编码，池化策略由 BERT 内部控制，此处算子主要服务旧版 Transformer 文本路径与知识编码中的可选序列聚合。

4.1.2 文本编码器（BERT Text Encoder）

功能

- 使用预训练 BERT 直接生成文本序列表示与句子级向量 p。

- 所有上游 tokenization 与 masking 均由 Batcher 完成。

关键特性

1. **严格对齐 KAN 的 D（hidden_size）**：避免论文时代嵌入规模不统一问题。
2. **三种池化策略**：cls / mean / mean+mask，满足不同下游需求。
3. **可冻结参数（freeze_encoder）**，便于小数据集训练稳定化。

输出

- sequence_output：形状 (B, L, D)
- pooled_output：形状 (B, D)，即新闻表示 p

4.1.3 Transformer 编码器（TransformerEncoder）

用于编码实体与实体上下文序列。具备三种位置编码：

1. **Sinusoidal**（原论文兼容）
2. **RoPE（Rotary Position Embedding）** —— 本工程默认，用于更现代的 Transformer 设计
3. **None** —— 适用于显式位置对齐不重要的场景

RoPE 的实现基于两个自定义算子：

- _RoPEEncoderLayer
- _RoPETransformerEncoder

只对 **Q/K** 应用旋转位置编码，从而增强位置感知能力，符合当前大模型架构趋势。

4.1.4 知识编码器（KnowledgeEncoder）

该模块接收实体嵌入与上下文嵌入，并输出对应的 Transformer 编码序列：

- q′ = Transformer(entity_embeddings)
- r′ = Transformer(context_embeddings)

若选择 share_encoder=True，两者将共享参数，可减少模型规模。

本工程中用法：

- 明确实体上下文序列长度一致
- 利用 padding mask 保持序列对齐
- 不执行池化（延后至注意力模块）

4.1.5 知识注意力模块（Knowledge-aware Attention）

提供两类注意力算子，是 KAN 的核心创新：

(1) N-E Attention（News → Entities）

矩阵形式：

$$q = \text{Attn}(p, q', q')$$

即使用新闻表示 p 作为 Query，实体编码 q′ 作为 K/V，得到加权实体表示 q。

意义：衡量“每个实体对新闻的重要性”。

(2) N-E²C Attention（News → (Entities, Contexts)）

矩阵形式：

$$r = \text{Attn}(p, q', r')$$

实体编码 q′ 作为 Key，实体上下文编码 r′ 作为 Value。

意义：通过实体的重要性分布选择对“上下文知识”加权，实现 **实体驱动的上下文选择机制**。

工程实现中二者接口统一，均支持：

- batch_first=True
- 返回 attention weights（用于调试/可视化）

4.2 主模型设计（KAN 主网络）

KAN 主模型整合了文本、知识与注意力模块，构成最终可训练的分类网络。

4.2.1 初始化阶段：模块注入与维度对齐

三大关键过程：

1. **注入 BERT 文本编码器**
2. **依据 BERT hidden size 自动对齐：**
    - knowledge.encoder.d_model = D

- attention.d_model = D

  这保证所有注意力与 Transformer 共享同一向量空间

3. **构建知识注意力模块，实现 q 与 r 的生成**

特殊工程增强

- 若实体嵌入维度不匹配，将生成警告，但保持兼容性不崩溃。
- 提供 zero_if_no_entities 机制，使模型在无实体输入时退化为纯文本模型，提高鲁棒性。

4.2.2 前向传播（Forward）

Step 1：新闻表示 p

$$p = \text{BERT}(token\_ids)$$

Step 2：实体编码与上下文编码

$$(q', r') = \text{KnowledgeEncoder}(entity\_emb, ctx\_emb)$$

Step 3：知识注意力融合

$$q = \text{N–E}(p, q'), r = \text{N–E}^2\text{C}(p, q', r')$$

若无实体：

$$q = r = 0$$

（由配置 zero_if_no_entities 控制）

Step 4：特征拼接

$$z = [p;\ q;\ r]$$

Step 5：分类器

$$\text{logits} = \text{Linear}(\text{Dropout}(z))$$

输出形状为 $(B, num_{classes})$。

5. kan.training

本节旨在系统化阐述 KAN 框架中训练与评估子系统（kan.training）的架构设计、算子组织方式以及工程层面的可复现性策略。我们重点关注数据在训练阶段的流动方式、算子之间的组合关系、梯度更新机制、学习率调度策略以及 checkpoint/metrics 工程化处理方式。所有内容均以统一配置体系为中心，围绕 TrainingConfig 与 EvaluationConfig 的结构化参数展开分析。

5.1 训练阶段的算子组织逻辑

5.1.1 输入数据形态与 BatchEncoding 统一接口

KAN 的训练阶段严格依赖 BatchEncoding 数据结构，它是 kan.repr.batching 输出的规范化批处理格式。其内部包含：

- token_ids 与 token_padding_mask：文本侧输入；
- entity_ids、entity_padding_mask：实体侧输入；
- context_ids、context_padding_mask：上下文实体输入；
- labels：监督信号；
- ids：样本编号，用于日志跟踪；

Trainer 通过 _batch_to_device() 将该结构整体搬迁至目标训练设备，实现批处理结构的稳定传递。这种设计保证了 KAN 模型可以通过单一入口 model(batch_encoding) 完成全图前向传播，避免了多路参数手动管理的复杂性。

这种统一的 Batch 接口使得训练流水线具备两个工程优势：

1. **算子解耦**：模型结构不需要关心输入的来源（词表模式/预训练 BERT 模式），trainer 也不需要猜测模型输入类型。

2. **可维护性增强**：如需扩展输入模态（如多模态图像流），可通过扩展 BatchEncoding 而非更改训练器逻辑。

5.1.2 前向传播与多分支输出的工程处理

训练器允许模型返回三类输出结构：

1. Tensor（直接视为 logits）

2. (logits, aux) 元组（忽略辅助项）

3. {"logits": Tensor, ...} 映射结构

这种多分支处理机制确保模型实现可灵活插入额外的辅助输出（如注意力可视化、知识利用统计等），而不影响训练主路径。这一设计使得模型与训练器之间保持松耦合，特别适合研究型代码的快速迭代。

5.1.3 损失函数与二分类任务的 logit 规范化

训练流程采用：

- **BCEWithLogitsLoss**：适用于 fake news（正类为 1）预测；

- logits 自动处理规则：

- 若模型输出形状为 (B, 2)，取第二列作为正类 logit；
- 若为 (B, 1) 或 (B,)，直接使用。

这是一个重要的工程决策：即使模型内部采用二分类 softmax 输出，训练器仍能稳健提取正类对数几率，保持训练统一性（）。

这种自动分发机制有效避免了“模型输出形态改变导致训练崩溃”的常见问题，特别有利于 KAN 的多编码器结构不断迭代的研究场景。

5.1.4 区分 warmup 的学习率调度策略

训练器实现了一个简化但稳健的线性 warmup 调度器：

- 在 warmup 期间：

$$\text{lr} = \text{base\_lr} \cdot \frac{\text{step}}{\text{warmup\_steps}}$$

- warmup 完成后：
  直接回落到 base_lr，除非用户通过 TrainingConfig 声明更复杂调度器。

这种设计的核心考虑是：

1. **避免不必要的复杂度**（工程简化原则）。
2. **warmup 足以稳定 Transformer/BERT 训练前期的梯度波动**。
3. 若使用余弦衰减或多项式衰减，配置接口已预留，后期通过扩展 _update_learning_rate() 即可支持。

5.1.5 梯度裁剪与稳定性保证

模型参数更新前，通过 nn.utils.clip_grad_norm_ 对梯度范数进行裁剪。此策略是深度 Transformer 训练中的标准实践，可有效抑制梯度爆炸，特别是在使用两路 Transformer（文本与知识编码器）并行反向传播时尤为必要。

这种稳定性保证在小数据集（如本任务中的 3673 条样本）上显得尤为重要。

5.1.6 Checkpoint 的结构化持久化

训练器每个 epoch 根据配置保存 checkpoint，包含：

- model_state_dict
- training_config（保留实验信息）
- epoch（恢复训练用）

这种结构化持久化便于：

1. 部署推理时从任意 epoch 加载；
2. 实验可复现性（再现模型性能）；
3. 后续基于中间 epoch 进行微调尝试。

文件结构遵循标准 PyTorch 保存格式，兼容 evaluator 加载逻辑。

5.2 可复现性的系统化保证

为了确保训练结果的可复现性，系统采用多层随机性控制：

1. 统一设置 Python random 种子；

2. NumPy 随机性设定；

3. PyTorch CPU/GPU 随机性设定；

4. 强制 cuDNN 进入 deterministic 模式。

这种多层次随机性屏蔽，使得：

- KAN 在 CPU/GPU 环境之间具备较强一致性；
- 实验复现实验流程具备高度稳定性；
- CLI 训练与 notebook 调试之间结果一致。

5.3 评估流水线：推理、指标与结果输出

Evaluator 作为一个独立模块，具备以下设计目标：

5.3.1 推理接口的通用性

评估阶段接受任意 mapping 作为 batch，包括：

- "id" 或 "ids"
- "labels"（可选）
- 任意模型输入字段

并自动调用 model(**inputs)
这种设计与训练器的 BatchEncoding 松耦合，大幅提升灵活性，便于将模型嵌入外部系统（如 flask API、比赛测试脚本）。

5.3.2 预测概率生成与一致化逻辑

评估器统一将 logits 转为概率：

$$\text{prob} = \sigma(\text{logit})$$

确保：

- 推理输出是 0 到 1 的概率；
- 无论模型内部使用何种输出结构，评估器都能自动兼容（）。

5.3.3 指标计算与容错性

评估器提供：

- Accuracy
- Precision（正类）
- Recall（正类）
- F1-score
- ROC-AUC（若单类，不报错而返回 None）

核心工程点：

1. 即使模型输出全 0 或全 1，也不会因 AUC 不可定义而崩溃；
2. 指标以结构化 dataclass 输出，便于日志与可视化（）。

5.3.4 CSV 输出格式的稳定保证

评估器将 (id, prob) 统一输出，其格式与平台完全对齐，避免用户手动实现 CSV 导出可能带来的格式错误（如 index 列、prob 命名错误）。

写入逻辑通过 write_probability_csv() 保证两列严格匹配。

5.4 训练—评估之间的整体架构协同

从系统角度，Trainer 与 Evaluator 的设计形成以下闭环：

1. **训练器输出 checkpoint**，格式稳定；
2. **评估器自动加载 checkpoint**，无需匹配额外上下文；
3. **统一配置体系（ExperimentConfig）驱动两个模块**（）；
4. **随机性控制模块贯穿训练与推理**。

这种高度一致的工程化设计确保：

- 训练 → 推理 → 结果输出的全链路稳健性；
- 新模型结构（如加入 BERT）可在最小改动下集成；
- 开发者无需在训练与评估阶段重复处理输入数据逻辑。

6. kan.utils

本节介绍 KAN 系统的底层通用支持库 **kan.utils**，其职责是为整个端到端的训练与推理流程提供一致、可靠、可复现且工业级的基础设施。与模型、数据处理与训练管线不同，kan.utils 不涉及任何语义建模或神经网络算子，而是通过对 日志（logging）、随机性控制（reproducibility）、评价指标（metrics） 与 配置解析

（configuration） 等底层组件的统一封装，构建稳健的工程基座。此类工具模块是任何可运行的大规模学习系统的基础，使得 KAN 不仅是论文实现代码，更是具备工程可维护性与可扩展性的体系。

整体而言，kan.utils 在数据流管线之外独立存在，但贯穿模型开发的每一个阶段：从预处理、批处理、训练监控到最终评估。其核心特征包括：

（1）保持所有模块间的**一致性行为**（例如统一日志格式）；

（2）保证训练实验的**可复现性**；

（3）提供规范化、对齐工业实践的**指标输出与结果存档**；

（4）通过集中式配置解析减少系统间耦合。

下文将从四个子模块展开介绍。

6.1 日志系统（kan.utils.logging）

日志系统是 KAN 工程层最重要的底座之一。该模块实现了一个 **统一、无重复配置（idempotent）、微秒级精度** 的日志记录器，用以满足训练阶段对高时间分辨率的需求，并显式区分标准输出与错误输出两类信息。

6.1.1 设计目标

1. **跨平台稳定性**：解决 Windows 控制台的默认 GBK 编码可能导致的 UnicodeEncodeError，因此增加了 UTF-8 强制重配置机制。

2. **信息分流（stdout vs stderr）**：

    - INFO 级别输出用于正常训练进度展示，重定向到 stdout；

- WARNING/ERROR 则输出到 stderr，以便 CLI 与脚本能依靠管道区分错误类型。

3. **格式统一性：**所有日志均采用统一的无空格微秒时间戳格式，以提升可读性与可检索性。

6.1.2 关键实现

日志器通过 _ExactLevelFilter 限制 INFO 信息仅投递到 stdout，避免常见的双重输出问题。核心格式定义如下：

[YYYY-MM-DD-HH:MM:SS.microseconds] LEVEL @{logger}: message

这一显式、可追踪格式对于训练中分析性能瓶颈、监控 epoch 流程尤为关键。

6.2 随机性控制（kan.utils.seed）

深度学习实验重现性一直是研究与工程实践中的挑战。KAN 的 set_global_seed() 将 Python、NumPy 与 PyTorch（含 CUDA）多个随机源统一在一个入口，严格控制每次实验的随机数状态。

该模块提供：

- 多源随机种子同步
- GPU 随机状态设定（torch.cuda.manual_seed_all）
- cuDNN 的 deterministic 模式选择
- 对潜在系统级不一致性提供显式注释与解释

在 KAN 中，词表构建、实体链接顺序、batch shuffle、Transformer 的 dropout 与数据加载线程均会引入随机性。因此，统一的随机种子机制对于：

- 多次训练结果的稳定性
- 初步调参阶段的可复现性

具有核心意义。

6.3 指标计算与结果导出（kan.utils.metrics）

该模块提供了 KAN 项目统一的二分类指标库，严格对齐 Fake News Detection 中的标准指标（Accuracy、Precision、Recall、F1、AUC）。此外，它还包含预测概率的存档工具，保证工程端结果输出格式的一致性。

6.3.1 指标容器设计

BinaryClassificationMetrics 使用 dataclass 封装所有指标，使结果具备：

- 结构化（structured）
- 可序列化（JSON/log-friendly）
- 与 logger 兼容的格式化能力

这种设计避免了在训练与评估模块中重复拼接输出字符串，提高可靠性。

6.3.2 核心算子

compute_binary_classification_metrics() 统一了输入格式（tensor、list、numpy），使用 sklearn 的成熟实现计算指标。
其中 AUC 在样本类别不平衡情况下可能无法计算，因此模块实现了容错逻辑并记录警告，这保证了系统面对极端数据集时不会崩溃。

6.4 配置管理（kan.utils.configs）

配置系统作为 KAN 工程的总控制入口，负责将 JSON 格式的实验配置文件解析为各子模块的 dataclass 实例。其设计核心是**配置与模块实现的弱耦合**：模型组件不会直接读取 JSON，而是依赖 dataclass，增强了类型安全性与可维护性。

6.4.1 JSON → dataclass 的解析流程

该模块完成以下任务：

1. 层次化加载数据、预处理、词表、嵌入、Transformer 编码器、注意力模块、训练配置等多个子系统；
2. 通过 _filter_kwargs 自动舍弃未知字段，避免因 JSON 配置更新产生崩溃；
3. 为缺省字段注入合理默认值（例如 embedding 的 vocab_size=0 作为占位符）；
4. 支持 BERT 与非 BERT 模式共存。

在整个训练流程中，配置系统确保：

- **可复现性**：每一次训练均由配置精确定义
- **可组合性**：不同组件的配置彼此独立，可灵活重组
- **可扩展性**：新增模型或编码器不需修改核心加载逻辑

从工程角度，配置系统是项目可维护性的关键。

五、命令行前端与微内核

1. 设计动机

在本项目中，命令行前端（CLI）与核心运行时（ExperimentRuntime）的设计采用了显式解耦的工程范式，其根本动机来自两个方面。

首先，虚假新闻检测模型的训练、评估与推理过程均依赖复杂的内部子系统，包括数据预处理、知识图谱查询缓存、词表构建、批处理管线以及多分支知识增强模型（KAN）本身。若将这些逻辑直接绑定于命令行接口，不仅会大幅抬升前端复杂度，也会降低可测试性与可维护性。因此，我们希望为最终使用者提供一个高度抽象化、最小侵入性的交互层，使用户在不理解内部系统细节的前提下即可完成训练与预测。

其次，KAN 系统在重构后呈现高度模块化的结构，嵌入模块、注意力机制、批处理器、实体上下文构造器等均可能在后续迭代中替换或扩展。为了保证这些模块的可插拔性与演进能力，需要一个稳定的调度与资源管理框架，使模型或训练管线的修改不会传递到 CLI 层。基于此考虑，我们采用“微内核式（micro-kernel）”运行时，将核心的资源管理、生命周期控制、任务调度从业务逻辑中抽离，从而确保结构稳定性与扩展性。

综上，本节所讨论的 CLI–Runtime 解耦体系在本项目中不仅是工程实现的基础框架，同时也是我们实现本地化、工业化和可持续演进能力的关键技术支点。

2. 命令行前端的架构设计

命令行前端（kan_cli）被设计为一个极薄的壳层（thin wrapper），其唯一职责包括：

(1) **解析用户输入的命令行参数**；

(2) **根据解析结果选择并调度合适的系统任务**。

2.1 子命令结构

CLI 采用分层命令体系，将不同实验流程封装为独立子命令，包括：

- train：执行训练流程，包括数据加载、词表构建、模型训练与 checkpoint 保存。
- evaluate：在验证集或测试集上评估模型精度、召回率等指标。
- predict：对未标注样本进行推理，并输出标准化预测结果文件。

每个子命令均仅负责参数声明，不包含任何业务逻辑，例如模型构建、数据预处理、损失计算等均不在 CLI 范畴之内。这样可保持 CLI 的稳定性，使其在模型变化、训练方式演进或数据处理逻辑更新时无需改动。

2.2 全局配置加载流程

CLI 统一使用 --config 指定的 JSON 配置文件，并允许命令行覆盖部分关键参数（如设备类型、批大小），从而实现灵活复现与配置管理。所有配置加载任务被委托给运行时处理，进一步减少 CLI 的复杂度。

2.3 前端无业务逻辑原则

我们将“CLI 不包含业务逻辑”作为明确的工程约束，理由包括：

- 业务逻辑变动频繁，而 CLI 需要保持 API 稳定；
- 多任务共享资源，资源管理应交由运行时统一处理；
- 任务本身需要可测试性，而 CLI 代码不易进行单元测试；
- 模型或数据模块的扩展不得牵连前端。

该设计保障了 CLI 的长期可维护性，使其可以在整个项目生命周期内保持稳定。

3. 微内核运行时模型

ExperimentRuntime 是系统的核心调度器，其设计灵感来源于操作系统的微内核结构。运行时本身不承担业务逻辑，而是 **资源与生命周期管理器（Resource & Lifecycle Manager）**。

3.1 资源与生命周期管理

运行时通过一系列 lazy-initialized 接口提供：

- 配置管理
- 数据预处理管线与 DataLoader
- 词表构建与持久化
- 模型构建与加载
- 优化器、调度器与训练状态
- 日志与输出目录管理

这些资源均在运行时内部进行统一生命周期管理，而不会泄漏到 CLI 中。

3.2 RuntimeState 有限状态机

运行时通过状态机确保任务执行有序且可追踪，典型状态如下：

INIT → READY → RUNNING → (COMPLETED | FAILED)

每个任务在进入 RUNNING 前必须完成必要的初始化（例如数据预处理），而任务结束后必须转移至终止态。此机制降低了工程错误（如重复初始化、资源竞态）的可能性。

3.3 Lazy Construction 原则

为了提升性能与用户体验，运行时严格遵循 lazy-init 原则，例如：

- 在训练前才构建词表
- 在推理前才加载模型权重
- 在首次访问时才初始化 DataLoader

取消不必要的启动开销，使系统在处理小数据集、调试阶段或快速迭代时更加轻量。

3.4 微内核的工程意义

该体系类似于操作系统的微内核：

操作系统概念	KAN 系统对应模块
微内核（kernel）	ExperimentRuntime
用户态程序	CLI 与 Task
驱动 / 服务	TrainTask / EvaluateTask / PredictTask

运行时提供最小功能集合：调度、资源管理和生命周期控制；而所有“服务”均由任务插件实现，从而实现高度可扩展与长期稳定。

4. 任务系统（TaskRegistry 与 ExperimentTask）

任务系统构成了 CLI 与 Runtime 之间的关键桥梁，其目的是让任务以“插件化”方式注入运行时，而非硬编码。

4.1 注册表模式（Registry Pattern）

系统通过 TaskRegistry 保存任务名到任务类的映射。
CLI 仅传递字符串任务名，如 "train"，由运行时通过注册表解析对应任务类。

该设计实现了：

- 任务查找过程与任务实现逻辑解耦
- 去除 CLI 与具体任务类的 import 依赖
- 多任务共存而互不干扰

4.2 ExperimentTask 抽象接口

所有任务均继承 ExperimentTask，并实现统一的 run() 方法。
典型任务包括：

- TrainTask
- EvaluateTask
- PredictTask

每个任务均通过 self.runtime 获取统一资源服务，确保执行逻辑独立于运行时的内部实现细节。

4.3 可扩展性展示

若新增任务（如 ErrorAnalysisTask 或 ExportEmbeddingTask），开发者只需：

1. 实现一个继承 ExperimentTask 的新类
2. 使用 @register_task("xxx") 装饰器注册

无需修改 CLI 与 Runtime，这体现了工程体系的可组合性与长期演进能力。

5. 分层设计的工程效果

5.1 长期维护性与解耦

CLI 与核心逻辑完全隔离，使系统在以下变动中保持稳定：

- 模型实现更替（如替换文本编码器为 BERT）
- 批处理结构变更
- 词表构建策略变化
- 新的训练策略或调度器引入

所有变动均局限于内部模块与任务层。

5.2 扩展性

添加新任务无需修改运行时，也无需触及 CLI。
这种插件式设计极适合研究系统快速迭代与工业系统持续演进。

5.3 明确抽象边界

通过“前端–微内核–任务”的三级结构，系统边界清晰，避免逻辑乱入：

- CLI 不处理模型
- Runtime 不实现业务
- Task 不管理资源
- 库模块不关心执行条件

六、不足之处

1. 语义空间不一致与小数据集条件下的对齐难题

1.1 引入 BERT 导致的语义空间断裂

KAN 原始结构中，文本、实体、实体上下文均通过结构相同、规模相近的编码器得到表示。这意味着三类向量同处于同一训练过程、同一表达空间，其相似度计算具有天然一致性【】。

然而，当文本编码器被替换为 BERT 时，该一致性被破坏。原因包括：

（1）预训练模型具有高度结构化的语义流形（manifold）

BERT 的表示是基于大规模语料训练得到的成熟语义空间，其几何结构、局部邻域关系与语言统计规律高度一致。而实体与上下文若仍采用：

- 随机初始化的嵌入
- 小模型 Transformer

- 或简单邻居平均表示

其产生的向量空间与 BERT 输出空间之间将不存在天然同构关系。

因此，跨空间比较（例如 N-E Attention 中的 Attn(p, q')）缺乏共同几何基础，注意力机制难以稳定衡量“新闻对实体的语义相似度”。

（2）注意力机制本质依赖空间可比性

多头注意力依赖 $QK^{\top}$反映语义相似度，但当：

- Q来自 BERT 的上下文化表征
- K、V 来自浅层或随机初始化模型

则内积不再衡量语义相近程度，只反映空间尺度和分布差异，进而使得注意力权重失去解释性。

（3）对齐需要额外监督，而分类任务本身不足以构成对齐信号

理论上，可以通过额外损失使空间对齐，例如：

- 显式投影层（linear projection）
- 对比学习（contrastive alignment）
- 基于实体描述文本的联合编码

但 KAN 结构中不存在这些额外监督，其唯一目标是真假分类。
真假标签仅作用于最终预测，不对“文本空间—实体空间”的对应关系提供足够约束。

因此，引入 BERT 在理论上必然导致空间不一致问题，而 KAN 原架构未设计任何克服此问题的机制。

1.2 不引入 BERT 时，小数据训练无法获得足够语义表达力

如果为了保持空间一致性而放弃 BERT，则文本、实体、上下文均由小规模 Transformer 从头训练。在小规模数据条件下，这一做法存在结构性限制：

（1）语料规模小于训练语义模型的基本要求

KAN 原文数据平均长度为 700–1400 字，实体密度高、信息多。但在微型中文微博场景下：

- 文本长度显著缩短
- 主题分散
- 噪声高、词分布稀疏

这类语料不足以支持训练出具有稳定语义能力的 Transformer Encoder。

（2）没有足够数据学习上下文相关的表达

Transformer 的优势在于捕获长距离依赖与上下文语义，但这依赖大量训练实例才能使注意力权重形成可泛化的模式。小规模数据下，模型难以形成：

- 稳定的词序特征
- 可靠的语义邻域结构
- 可用于为实体提供语义支撑的文本表征

因此，即便三类编码器的空间是一致的，其表达力仍然受限。

（3）弱文本表达力会连带削弱实体与上下文信号

KAN 的设计依赖文本向量作为 Query 去选择重要实体（N-E attention）及其上下文（N-E2C attention）。若文本向量本身表达能力不足，则注意力无法准确评估实体的重要性，整体架构效果受限。

因此，不引入 BERT 虽然解决了空间不一致问题，但会带来语义建模能力不足的根本限制。

1.3 小数据下「空间一致性」与「表达能力」无法同时满足

以上两点形成一个结构性矛盾：

引入 BERT → 表达力强，但空间不一致难以弥补

不引入 BERT → 空间一致，但表达力不足以支撑任务**

此矛盾的根本原因在于：

- 预训练语言模型的空间与从随机初始化训练的空间天然不同；
- 单一二分类任务无法提供跨空间对齐所需的监督信号；
- 小数据无法训练出有足够表达力的统一编码器。

因此，该问题并非完全由实现细节导致，而是架构层面在“小数据 + 外部实体 + 跨模型表示”的组合下的理论瓶颈。

2. 强模型与小数据集的张力

2.1 模型容量的增长速度远快于数据规模

在知识增强架构中引入预训练语言模型（如 BERT）会显著提高文本编码器的表示能力。从模型复杂度视角来看，其容量提升主要体现在：

1. 参数量级从百万级上升到亿级；
2. 模型的函数空间（hypothesis space）呈指数级扩展；
3. 表达能力与泛化能力远高于随机初始化的小型模型。

然而，小数据集在训练中能提供的约束是有限的。对每一个样本，训练给模型提供的实际监督仅来自：

- 文本序列的局部词序统计；
- 实体集合（若存在）；
- 单一真假标签。

因此，当模型容量远大于数据变量的多样性时，学习过程不可避免地面临“欠约束（under-constrained）”的问题，即：

模型的可表达空间大幅扩展，但数据提供的约束信号并未同步增长。

这构成了强模型与小数据之间的第一类张力。

2.2 高容量模型会放大数据噪声而非吸收结构信息

高容量模型具有学习复杂模式的能力，但在小数据条件下，被学习的模式往往不是任务相关结构，而是数据中的偶然性（偶发分布、表达习惯噪声等）。
理论上，这可视为：

- 信噪比（Signal-to-Noise Ratio）过低；
- 模型可以拟合噪声，却无法从噪声中辨识信号。

这对 KAN 架构尤其重要，因为外部知识模块依赖**稳定的文本表示**作为 Query，而 Query 若包含大量噪声成分，会直接影响：

- N-E attention 的实体选择；
- N-E2C attention 的上下文权重分配；
- 下游分类器的决策边界。

因此，高容量编码器可能在小数据条件下学习到“局部片段特有模式”，而不是“新闻语义结构”，进而削弱知识融合模块的有效性。

2.3 强模型的参数自由度扩大了跨模态对齐的难度

KAN 结构本质上是一个多路输入的架构：

- 文本（高维、序列化、上下文化）
- 实体（符号化、稀疏、KG 结构驱动）
- 实体上下文（邻居平均、非序列化）

在原论文中三路编码器的容量大体一致，因此三类表示可以同时被梯度信号推动向同一语义空间靠拢。然而：

当文本编码能力被大幅增强时，其表示空间的复杂性被整体拉高，而实体与上下文两侧的编码器仍保持低容量。从理论角度看，这会导致：

1. 文本表示对“语义维度”的分辨率提高；
2. 实体表示仍停留在“粗粒度词义或邻居平均”层面；
3. 注意力机制在两个量级的表示之间难以寻找线性可比的特征维度。

换言之，模型容量的不对称使得跨模态注意力成为高维向量与低维近似之间的匹配问题，而匹配空间中可能不存在稳定的最优点。

因此，强模型不仅放大了文本表征能力，也放大了**对齐问题本身的规模**。

2.4 小数据集不足以提供结构性监督

无论是否引入外部知识，假新闻检测本质上是一个二分类任务。
其监督信号来自类别标签，而类别标签本身无法提供：

- 文本语义对齐监督；
- 文本—实体映射监督；
- 实体—上下文结构监督。

也就是说，数据集规模限制了监督质量，而监督质量决定了模型是否能够学习跨表示空间的结构耦合。

在小数据集情境下：

监督约束的数量与模型自由度的数量不成比例，模型无法学到“跨模态结构规律”，只能学习到“局部数据经验”。

这会使强模型难以在理论上形成稳定的知识增强机制。

2.5 小数据下强模型的欠约束与知识增强的过要求形成矛盾

知识增强模型天生对以下三点有强依赖：

1. 文本编码器必须具备较强语义能力
2. 实体编码器必须与文本空间兼容
3. 注意力机制必须能从表达结构中提取跨模态关联

然而，小数据集无法同时为三者提供足够约束。于是结构性矛盾出现：

- 若提升文本编码器，表示空间不一致；
- 若保持三路一致，文本语义能力不足；
- 若加入额外对齐机制，监督又不足以训练它。

这意味着：

强模型要求高容量与高一致性，而小数据只能提供低自由度与弱监督，两者在结构上难以同时满足。

3. 知识图谱能力的局限性与工程替代实现的不足

3.1 完整实体链接系统的缺失导致知识输入质量下降

实体链接（Entity Linking, EL）通常包括两个子任务：

（1）Mention Detection（实体片段识别）

TagMe 使用广谱 NER、统计特征和 Wikipedia anchor 进行高召回率的 mention 检测。而我们当前的实现依赖：

- 分词结果；
- 基于 token 的逐词搜索；
- 缺乏 phrase-level matching 和 alias 模型。

理论上，这会导致：

- 大量实体 mentions 未被检测（低召回）；
- 误将普通词汇映射为实体（低精度）；
- 中文特有现象（缺词界、专名多样化）无建模机制。

（2）Entity Disambiguation（实体消歧）

TagMe 的核心优势在于利用：

- 全局 coherence；
- Wikipedia link graph；
- mention 上下文共现概率；
- prior popularity（先验概率）。

而我们现有方法仅依赖：

- Wikidata 的 wbsearchentities API 搜索结果；

- 排序策略依赖 API 返回顺序；
- 无上下文得分、无语义消歧、无知识一致性约束。

因此，从理论上可断定：

我们的 EL 信号是噪声远大于 TagMe 环境中假设的 EL 信号质量。

这使得 KAN 的“知识输入”与论文假设的“准确实体集”存在根本偏差。

3.2 缺乏 mention-level 判断意味着注意力机制无法得到预期监督

KAN 的 N-E attention 假设文本向量 p 能与实体编码 q′ 产生意义明确的语义相似度，从而得到实体重要性权重 α_i。但这一机制隐含一个前提：

实体集合 E 应当尽可能接近真实的 mentions 对应的实体集合。

而当知识图谱侧的实体集具有偏差时，理论问题随之出现：

1. 注意力计算被迫在“错误候选实体”间分布权重；
2. 中间编码 q′ 不再是语义可比对象，而是噪声实体的随机 embedding；
3. N-E2C attention 进一步传播噪声，使上下文表示 r′ 偏离真实语义结构；
4. 融合时 $p \oplus q \oplus r$ 的语义一致性被破坏。

换言之，**实体链接的不成熟会从输入端破坏整个知识增强路径的梯度传播逻辑**。

3.3 简化的实体上下文抽取无法复现论文中的 KG 结构信息

论文中，实体上下文 ec(e) 是实体所有一跳邻居，其语义作用在于：

- 提供实体的语义类别（如人/地点/组织）；

- 提供实体的关系模式（如政治人物、地理隶属关系等）；
- 作为实体 disambiguation 的补充特征。

然而中文 Wikidata 查询存在以下结构性差异：

（1）中文 label 稀疏、描述不一致

许多 Wikidata 实体缺乏高质量中文标签，导致搜索匹配不稳定。

（2）邻居语义分布不均衡

一些常见实体（如政治人物、城市）拥有数百个邻居，噪声极高。

（3）我们当前的 neighbor 查询使用通用 SPARQL，缺乏：

- relation importance 模型；
- property selection 策略；
- type-consistency 筛选；
- KG 上的路径过滤。

这意味着我们抽取的上下文更多是：

结构噪声的集合，而非论文假设的“语义相关的实体上下文”。

上下文表示 ec' 的作用会因此弱化，甚至干扰主模型。

3.4 自建 KG 客户端的工程限制导致鲁棒性不足

相比 TagMe 的工业级稳定性，我们的实现存在不可避免的理论工程限制：

（1）搜索完全依赖 Wikidata API 且无全局索引

无法使用：

- alias 数据库
- Wikipedia anchor frequencies
- 反向链接图
- precomputed priors

所有搜索依赖“API 当前返回的字符串匹配”。
这本质上是一个**表面匹配系统（surface-matching system）**，不是 EL 系统。

（2）SPARQL 查询的延迟、速率限制、超时影响整体 KG 信息质量

KG 模块的目的不是增加噪声，但网络延迟、并发限制会导致：

- 上下文不完整；
- 返回集随机性大；
- 缺失邻居被当作“真实结构”。

（3）缓存机制偏重工程性能，不提供语义正确性保证

缓存只能减少网络延迟，并不能提升实体链接精度。

4. 实体侧表示能力的局限性与引入强实体模型的潜在风险

4.1 实体嵌入过于基础难以承载丰富语义

从当前实现可见，我们的实体表示主要由三个组件构成：

（1）实体嵌入是随机初始化的表征

在 EntityEmbedding 中，实体 ID 被直接映射到可训练的向量表：

```
self.entity_embedding = nn.Embedding(
    num_embeddings=cfg.vocab_size,
    embedding_dim=cfg.d_model,
    padding_idx=cfg.padding_idx,
)
```

这意味着：

- 实体向量**不包含任何结构知识**（例如 KG 实体类别、关系模式）；
- 不包含**任何预训练语义**；
- 训练完全依赖当前任务的小数据集；
- 表示能力受到数据规模强烈限制。

在成熟 EL/KG 系统中，实体 embedding 通常来自：

- Wikipedia 预训练的 entity2vec；
- KG embedding 模型（TransE、DistMult、ComplEx、RotatE）；
- 基于实体描述文本的 Transformer 编码；
- 或联合训练的多模态特征。

相比之下，我们当前设计显著不成熟，这是受限于中文语料规模、KG 质量与工程可行性的结果。

（2）实体上下文表示是简单平均池化

上下文表示 ec(e) 在当前实现中由邻居实体 embedding 平均得到【】。这一做法：

- 不区分关系类型（predicate semantics）；
- 不考虑邻居的重要性（无 attention / weight）；
- 不保留结构特征（如一阶/二阶模式）；
- 对高阶噪声极为敏感。

因此，上下文向量更接近“邻居集合的稠密统计”，而非“实体语义的抽象”。
换言之，我们的实体上下文模块**远不能复现论文中假设的 KG 结构能力**。

（3）知识编码器是小规模 RoPE Transformer，容量有限

KnowledgeEncoder 使用 RoPE-based Transformer 对实体序列和上下文序列进行编码。尽管我们采用了 RoPE（结构上优于传统正弦位置编码），但：

- 编码层数通常为 1；
- 无预训练；
- 训练数据有限；
- 语义能力主要是“浅层序列变换”。

因此，实体/上下文 经 Transformer 编码后得到的 q′、r′ 本质上仍然是**弱语义表征**。

这一切共同决定：

我们的实体端管线整体处于非常基础的能力水平，与工业、研究实体表示系统相比差距显著。

4.2 引入高容量实体表示反而会加剧语义空间不一致

直觉上，增强实体 embedding（如使用 TransE/RotatE/BERT-based Entity Encoder）似乎能够改善实体表示贫弱的问题。但在当前架构下，这会产生新的理论风险。

（1）文本与实体编码器的容量进一步不对称

我们已经在前文说明：

文本侧使用 BERT → 表示能力远高于实体侧。

若实体侧也改用：

- 基于 KG 的预训练 embedding（TransE 类）
- 基于 Wikipedia 描述文本的 BERT 实体模型
- 多关系注意力实体模型（如 Deep Graph Infomax、GraphSAGE）

则实体侧表示空间将变成另一套**完全不同来源的高维语义流形**。

此时三路表示将成为：

- 文本：BERT 空间
- 实体：预训练 KG 空间
- 上下文：另一种 KG 空间或图结构空间

这些空间之间的几何性质、尺度、相似度分布、归一化方式均不同。

注意力机制要求：

Q 和 K 必须处于可比较空间，否则内积无法反映语义关系。

因此，引入强实体模型只会使当前已经存在的语义空间不一致问题更加严重。

（2）多种预训练空间之间缺乏统一参照系

BERT 的 embedding 空间由文本统计规律决定；
TransE 类 embedding 空间由 KG 的三元组几何约束决定；
GraphSAGE 等实体模型则由局部图结构决定。

这些空间之间没有天然映射关系。

若强行融合，会导致：

- 内积相似度失去可解释性；
- 注意力分布高度不稳定；
- 下游拼接后的向量表示不再具有统一尺度；
- 梯度传播路径复杂化、难以收敛。

换言之：

提高实体表示能力并不会自动提升知识增强效果，相反会破坏模型的整体语义一致性。

（3）强实体表示会把“弱监督对齐问题”进一步扩大

目前唯一的监督信号是二分类标签。若三路表示空间全部增强，参数规模成倍增长，而监督信号不变，问题会变成：

高维空间之间缺乏足够监督来学习对齐，模型自由度过高，负迁移风险加剧。

这将导致更严重的欠约束（under-constrained）与对齐失败（alignment failure）。

4.3 维持实体侧弱模型以控制跨空间差异

综合以上理论限制，我们当前的设计决策具有一定的合理性：

（1）实体侧保持轻量模型（线性 embedding + 小 Transformer）

目的是：

- 限制实体表示的自由度；
- 保持实体空间几何结构相对简单；
- 减少与文本 BERT 空间之间的差异幅度；
- 降低注意力机制的对齐压力。

（2）不引入预训练 KG embedding，避免新空间冲突

虽然实体表示较弱，但：

- 弱表示空间更容易被文本 Query 拉动；
- 注意力机制更容易形成稳定的权重结构；
- 训练过程不至于因空间不一致而震荡。

（3）保持实体与上下文共享 embedding 表（可选）

在我们的实现中支持共享 embedding（share_entity_context_embeddings=True），这样：

- 实体与上下文自然处于同一空间；
- 减少额外参数，提高稳定性；
- 避免实体与上下文之间出现独立的语义漂移。

这在弱监督小数据场景中是更安全、更稳健的设计。

七、总结与展望

本报告围绕 Knowledge-aware Attention Network（KAN）在中文场景下的工程化重构展开，系统分析了原论文方法在跨语言迁移、实体链接质量、知识图谱构造成本以及多编码器语义空间不一致等方面的局限，并基于实际任务需求给出了可运行、可复现实验流程的工程体系。

首先，在**中文本地化**方面，我们认为英文环境下的实体链接工具（如 TagMe）在中文语料上稳定性不足。为此，我们重新设计了本地化的实体抽取与知识接入流程，包括中文分词策略的调整、对 Wikidata 的缓存化与限速访问，以及对实体上下文的可控抽取机制，从而建立了一个稳定、可解释的中文 EL-KG pipeline。

其次，在**系统工程与工业可用性**方面，我们重构了数据处理、知识图谱客户端、vocab 管理、batching、训练器等关键模块，使得整个代码库具备一致的抽象层次、更强的可组合性、与配置驱动的工程规范。该体系克服了原论文实现未给出的细节（尤其是实体上下文池化、知识查询策略、batch 编排等），补足了研究到工程落地之间的缺口。

总体而言，本项目的价值并非在于算法创新，而在于将 KAN 的概念模型转换为适用于中文场景、可真实运行、可迭代优化的工程框架，为后续的知识增强型假新闻检测模型提供了一个可复用的基础设施。

尽管本报告已经建立了一个较为完整的中文 KAN 工程框架，但仍存在若干关键方向值得进一步研究：

（1）语义空间一致性的理论化与系统化解决

当前我们通过减轻实体侧编码器复杂度与引入 BERT 主干进行经验性缓解，但未从理论上刻画多空间对齐问题的本质。未来工作可探索：

- 基于对比学习（contrastive learning）的跨模态对齐机制
- 将实体、上下文、文本统一映射到共享空间的轻量投影
- 引入在小数据集上可稳定训练的正则化约束（如 canonical correlation constraints）

这些方法有望从根本上解决知识增强模型中长期存在的“强模型 × 小数据 × 多空间”三者之间的结构性矛盾。

（2）知识图谱质量与实体上下文的动态选择

目前采用 Wikidata 一跳邻域作为上下文，但其噪声大、语义覆盖不均匀、关系密度随实体而剧烈变化。未来可考虑：

- 构建任务特定的子图（task-specific KG）
- 通过关系类型加权、路径过滤等方式减少噪声邻域
- 在训练过程中动态选择实体上下文而非一次性静态抽取

从而提升知识输入的可控性与有效性。

（3）本地化实体链接进一步强化

尽管本报告提出了中文分词驱动的实体链接策略，但更强的实践方向包括：

- 引入深度 EL 模型（如 BLINK 的中文变体）
- 建立包含别名、缩写、歧义词的本地缓存式别名字典
- 在训练阶段利用实体表征与文本表征的相似度做在线重新对齐

使实体链接的召回与精度能够达到工业部署水平。

（4）面向大模型时代的轻量级知识增强范式

随着预训练大模型能力提升，显式知识图谱的必要性需重新评估。未来可探索：

- 结合 LLM 作为“软知识图谱”的生成器
- 比较显式 KG 与 LLM 隐式知识的互补性
- 构建 hybrid-KAN：由 KG 提供精确结构知识，大模型提供语义补充

从而寻找知识增强模型在未来更具性价比的形式。

参考文献或项目

Claude E. Shannon. 1948.

“A Mathematical Theory of Communication.”

Bell System Technical Journal, 27(3): 379–423; 27(4): 623–656.

https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Christopher M. Bishop. 2006.

Pattern Recognition and Machine Learning.

Springer.

https://link.springer.com/book/10.1007/978-0-387-45528-0

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones,

Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017.

“Attention Is All You Need.”

In *Advances in Neural Information Processing Systems (NeurIPS)*.

https://arxiv.org/abs/1706.03762

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019.

“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.”

In *NAACL-HLT 2019*, pp. 4171–4186.

https://arxiv.org/abs/1810.04805

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2021.

“RoFormer: Enhanced Transformer with Rotary Position Embedding.”

arXiv:2104.09864

https://arxiv.org/abs/2104.09864

Dan Hendrycks and Kevin Gimpel. 2016.
“Gaussian Error Linear Units (GELUs).”
arXiv:1606.08415
https://arxiv.org/abs/1606.08415

Ilya Loshchilov and Frank Hutter. 2017.
“SGDR: Stochastic Gradient Descent with Warm Restarts.”
In *ICLR 2017*.
https://arxiv.org/abs/1608.03983

Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. 2017.
“Fake News Detection on Social Media: A Data Mining Perspective.”
ACM SIGKDD Explorations Newsletter, 19(1): 22–36.
https://arxiv.org/abs/1708.01967

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2021.
“A Survey on Knowledge Graphs: Representation, Acquisition and Applications.”
IEEE Transactions on Neural Networks and Learning Systems.
https://arxiv.org/abs/2002.00388

Yaqian Dun, Kefei Tu, Chen Chen, Chunyan Hou, and Xiaojie Yuan. 2021.
“KAN: Knowledge-aware Attention Network for Fake News Detection.”
AAAI Conference on Artificial Intelligence, 35(1): 81–89.
https://ojs.aaai.org/index.php/AAAI/article/view/16085

Wanxiang Che, Zhenghua Li, and Ting Liu. 2010.
“LTP: A Chinese Language Technology Platform.”
In *COLING 2010 Demonstrations*, pp. 13–16.
https://aclanthology.org/C10-3003/

Junyi Sun.

Jieba Chinese text segmentation.

GitHub repository: https://github.com/fxsjy/jieba

Thomas Wolf et al. 2020.

“Transformers: State-of-the-Art Natural Language Processing.”

In *EMNLP 2020: System Demonstrations*, pp. 38–45.

https://aclanthology.org/2020.emnlp-demos.6/