慧言安鉴

2025
AIC

SenTox-GLDA：双编码语义解耦与多模态融合的网络暴力检测模型构建

参赛学校：[redacted]

成员：[redacted]

时间：2025.10

模型概述

框架核心思想在于并行建模情感与毒性特征，通过多尺度、层次化语义融合实现对网络暴力的高精度、实时化识别与预警。

目录

一、项目概述

1.1 项目背景与意义

随着互联网社交平台的快速发展，在线社区已成为现代数字社会中不可或缺的交流基础设施，为个体提供了前所未有的自我表达、信息传播与社交互动的机会。然而，这些益处正逐渐受到网络暴力内容的侵蚀，包括显性毒性言论以及隐性敌意表达，这不仅破坏平台秩序、扭曲公共舆论，还对用户心理健康造成潜在伤害。尤其在中文社交媒体中，由于语言特性及文化差异，隐性讽刺、同音替换、表情符号和特定俗语的广泛使用，使传统单文本分析或关键词过滤方法难以准确识别潜在滥用行为，导致治理效率低、响应滞后。

现有的网络暴力检测方法主要存在两类局限。一类是基于规则或关键词的简单方法，虽计算效率高，但难以捕捉细粒度情感线索和间接表达，如讽刺或同音词替代等。另一类是单流的深度学习模型（如 BERT、RoBERTa 或通用大语言模型），通常将毒性视为单一特征，忽略了情感倾向与恶意意图之间的复杂交互，因而在处理隐性、文化特定或语境依赖的滥用语言时表现不佳。例如，“急啦急啦，男宝又破防啦”这一评论，同时传递了敌意、讽刺和性别相关的隐性侮辱，如果仅依赖单一维度特征，模型难以有效捕捉其多层语义信息。

针对上述挑战，本项目提出一套基于多模态语义链的智能防控框架——SenTox-GLDA。框架核心思想在于并行建模情感与毒性特征，通过多尺度、层次化语义融合实现对网络暴力的高精度、实时化识别与预警。具体而言，我们设计了双编码器结构：BERT 用于情感特征提取，RoBERTa 用于毒性特征编码；随后通过自适应门控模块在 token 与序列层面动态融合特征，实现情感与毒性信号的互补与上下文敏感建模。为进一步捕捉局部滥用模式及篇章级攻击倾向，引入全局-局部差分注意力（GLDA）模块，局部差分注意力捕获短程词汇和句法线索，全局差分注意力建模讽刺、长程依赖及语境驱动的毒性。最终，采用 Kolmogorov–Arnold 网络（KAN）进行分类，相比传统全连接层，KAN 在提升非线性表达能力的同时降低计算开销，并增强模型解释性。此外，我们结合中文社交媒体特点设计了多元化数据增强策略，包括同音词替换、表情符号替换和句法结构扰动，以提升模型对隐性及对抗性滥用表达的鲁棒性。

通过整合情感与毒性线索、引入多尺度注意力以及轻量化高效的分类机制，SenTox-GLDA 能够更全面地理解中文网络语言中的复杂滥用行为，尤其适用于捕捉讽刺、反讽及情感隐性化的暴力内容，为社交平台提供更准确、可解释的智能防控能力。

1.2 赛题方向定位

本项目属于算法模型融合与应用方向。

整体方案基于 SenTox-GLDA 框架，构建了一种融合情感与毒性特征的双编码器架构，通过以下模型创新实现网络暴力内容的高精度识别：

1.多模态语义融合：模型采用双编码器结构进行情感与毒性特征提取。具体而言，BERT 基于 Weibo-100k 数据集微调以捕捉情感维度信息，而 RoBERTa 在经过表情符号增强的 ToxiCN 数据集上训练以提取毒性相关线索。两者输出通过自适应门控融合机制在 token 和序列级别进行加权整合，实现多维语义信息的互补与动态适应。融合后的特征被重构为结构化伪序列，以支持多尺度语义建模。

2.层次化注意力机制：提出全局-局部差分注意力（GLDA）模块，用于同时捕捉局部滥用模式与篇章级攻击信息。GLDA 由两部分组成：

①局部差分注意力（LDA）：利用滑动窗口差分增强对短程句法与词汇线索的敏感性，强化细粒度语义感知。

②全局差分注意力（GDA）：通过下采样注意力机制建模讽刺、上下文驱动的毒性及长程依赖，捕获跨句的语义关联。

3.轻量化分类结构：采用 Kolmogorov–Arnold 网络（KAN）替代传统全连接层，实现非线性决策边界的高效近似。KAN 通过共享基函数提高计算效率，并通过函数分解增强模型解释性，能够在情感-毒性融合空间内进行精确分类。

4.数据增强策略：结合中文社交媒体语言特点，设计多元化增强方法，包括同音词替换、表情符号替换以及句法结构扰动等，以提升模型对隐式及对抗性滥用表达的鲁棒性。

本方案旨在解决现有单文本静态模型在识别隐性、多模态网络暴力内容时的不足，为中文社交平台提供一套可扩展、高精度的实时防控系统。

二、算法模型设计

2.1 算法原理与架构

2.1.1 整体架构

我们提出 SenTox-GLDA 模型（图 1），这是一种双编码器架构，能够并行显式建模情感与毒性特征。该架构通过独立编码情感与毒性特征，更有效地捕捉

情感基调与恶意意图。这些表征通过自适应融合模块进行整合，该模块能根据上下文动态调整特征权重，从而生成更精准且上下文敏感的复合表征。

此外，我们引入全局-局部-差分注意力（GLDA）模块，同步捕获局部句法模式与全局语义趋势。在分类环节采用科尔莫戈罗夫-阿诺德网络（KANs），该网络能以较少参数实现强大的非线性建模能力，同时提升泛化性与运算效率。原始中文评论首先通过同义词替换、谐音置换、表情符号插入、结构扰动和回译等技术进行数据增强，以模拟多样化的语言变体。经处理的文本随后被分词，并分别通过情感编码器（捕捉情感极性）和毒性编码器（捕捉攻击性语义）进行并行编码。二者的隐层表示通过自适应融合模块进行组合，并经由全局-局部差分注意力（GLDA）机制进一步增强——该机制在语篇和词元层面整合了多频特征。最终，科尔莫戈罗夫-阿诺德网络（KAN）分类器生成可解释的预测结果，包括对地域、种族、性别等类别的细粒度判别。该技术代码已开源至 github 平台：*https://github.com/kanglzu/SenTox-GLDA*.

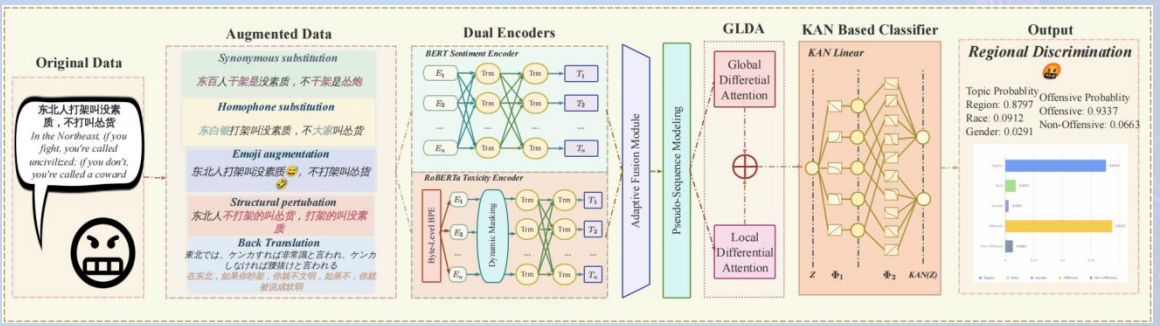

图 2.1：SenTox-GLDA 框架整体架构

2.1.2 模块设计

预训练双编码器框架

该研究提出的双编码器框架整合了两个预训练的 Transformer 模型，每个模型专门处理辱骂性语言的不同互补维度：

（1）基于 BERT-base-Chinese 的编码器在微博-100k 语料库上微调，用于提取情感表征。该数据集包含随意、富有表现力且情感充沛的用户生成文本，使该编码器在情感语调建模方面表现尤为出色。

（2）RoBERTa-wwm-ext-large 编码器在 Toxic-CN 数据集（结合 NMSL 项目的浅层和深层表情符号模式增强）上微调，专门捕捉毒性语义，包括中文社交媒体常见的俚语、表情符号驱动表达及辱骂性语言模式。

两个编码器均输出作为句子级抽象的[CLS]嵌入向量。RoBERTa 模型最终层的 1024 维[CLS]嵌入提供了毒性语义的高容量表征，其增强的维度与深层架构对于解析语境依赖性毒性（例如区分群体内部交流中的再生俚语与跨群体语境下的恶意使用）至关重要。这种细粒度建模使框架能够检测语义偏移现象——即表面中性的词汇根据文化或会话语境获得毒性含义。

BERT 情感编码器

情感分析模块采用微调的 BERT-base-Chinese 模型（图 2）从社交媒体文本中提取情感特征。BERT 转换器架构的核心是多头自注意力机制，该机制对词语间的上下文关系进行建模：

$$Attention(Q,K,V)=soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

其中 Q、K 和 V 分别代表查询矩阵、键矩阵和值矩阵，d_k表示维度。多头注意力机制让每个注意力头都能捕捉不同的语言依赖关系，实现对情感表达的分层上下文分析。模型优化是在微博-100k 数据集上使用交叉熵损失函数进行的:

$$\mathcal{L}=-\sum_{i=1}^{c} y_i \log(p_i)$$

微博-100k 数据集因其真实用户生成内容、自然情感表达、均衡的情感分布（59,993 条正面样本 vs 59,995 条负面样本）以及包含表情符号、网络俚语和非正式表达等复杂语言现象而被选用。这些特性带来了讽刺表达和隐性情感等现实场景挑战，使其成为微调模型的理想选择。

最终，编码器生成的 768 维[CLS]嵌入向量能以紧凑形式高效应地表征句子级情感内容。该表征特别适用于解决表层词汇信号与潜在情感意图相矛盾的歧义情况，从而为下游具备情感识别能力的滥用内容检测奠定坚实基础。

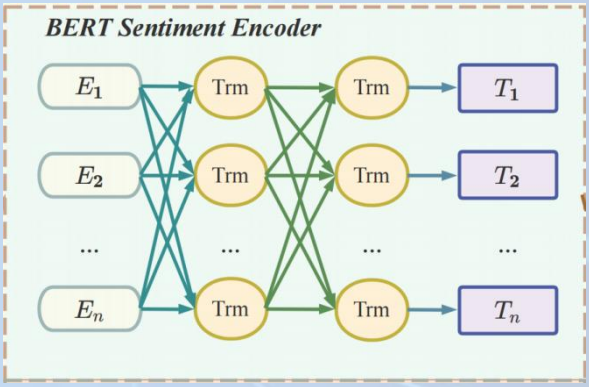

图 2.2：BERT 情感编码器

RoBERTa 毒性编码器

毒性检测模块采用经过优化的 RoBERTa-wwm-ext-large 模型（图 3），专门用于识别中文社交媒体文本中的有害内容。与标准 BERT 相比，RoBERTa 通过移除下一句预测目标并采用动态掩码技术增强预训练，从而生成更丰富的上下文表征，这对细致入微的毒性检测至关重要。

该模型架构包含 24 个 Transformer 层，具有 1024 维隐藏状态，通过多头自注意力机制处理文本序列。凭借 16 个注意力头，扩展的维度（d_k=1024）使模型能够捕捉与隐性辱骂相关的微妙词汇、句法和语用线索。分层结构逐步聚合从词级指标到句级毒性模式的信号，支持对网络隐性辱骂的细粒度判别。

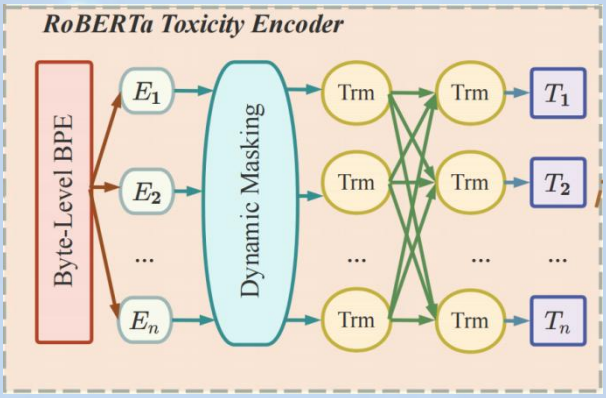

图 2.3：RoBERTa 毒性编码器

自适应融合模块

为有效利用情感与毒性表征，我们引入了一种能动态平衡各类特征贡献度的自适应融合模块。

设h_s和h_t分别表示来自 BERT 情感编码器和 RoBERTa 毒性编码器的[CLS]嵌入向量。融合表征h_f的计算公式为：

$$g = \sigma\left(W_g\left[h_s; h_t\right] + b_g\right)$$

$$h_f = g \cdot h_s + \left(1 - g\right) \cdot h_t$$

其中，$W_g \in \mathbb{R}^{d \times 2d}$，$b_g \in \mathbb{R}$，二者均为可训练参数；σ(·) 表示 Sigmoid 激活函数。门控向量 g 会通过学习，依据上下文对情感信号与毒性信号进行自适应加权，从而使模型在复杂细微的场景中能够突出情感线索，或在明确含有害内容的场景中优先关注毒性模式。

全局-局部-差分注意力机制

如图 4 所示，全局-局部-差分注意力（GLDA）机制通过联合捕捉序列中的局部变异和全局依赖关系，代表了相对于传统注意力模型的重大进步。与仅关注成对标记交互的标准注意力不同，GLDA 引入差分运算来显式建模语义的细微变化，从而增强对微妙模式的敏感性——这在情感分析和毒性检测等应用中至关重要，因为微小的语气变化就可能改变语义。

该模块首先将输入特征转换为伪序列，并通过池化操作提取多尺度表征。局部差分注意力（LDA）捕捉细粒度的标记级变化，而全局差分注意力（GDA）建模话语级依赖关系。二者的输出通过门控和投影动态整合，生成融合局部细节与全局上下文信息的表征。

该 GLDA 框架由两个互补组件构成：局部差分注意力（LDA）和全局差分注意力（GDA），将在 2.3.2、2.3.3 节中详细介绍

为实现信号融合，GLDA 还采用带有可学习权重系数 α 的动态融合机制，最终输出表现为局部特征与全局特征的加权组合。此外，残差连接的设计既保留了基线信息，又能防止处理过程中关键语义内容的丢失。

该框架的算法逻辑如算法 1 所示：其中 DiffAttn 计算差分注意力分数，P_{pool} 负责全局上下文提取的降采样处理。这种设计使 GLDA 能灵活适应不同长度和复杂度的序列在捕捉
数据微观层面变化与宏观结构时均保持高保真度。

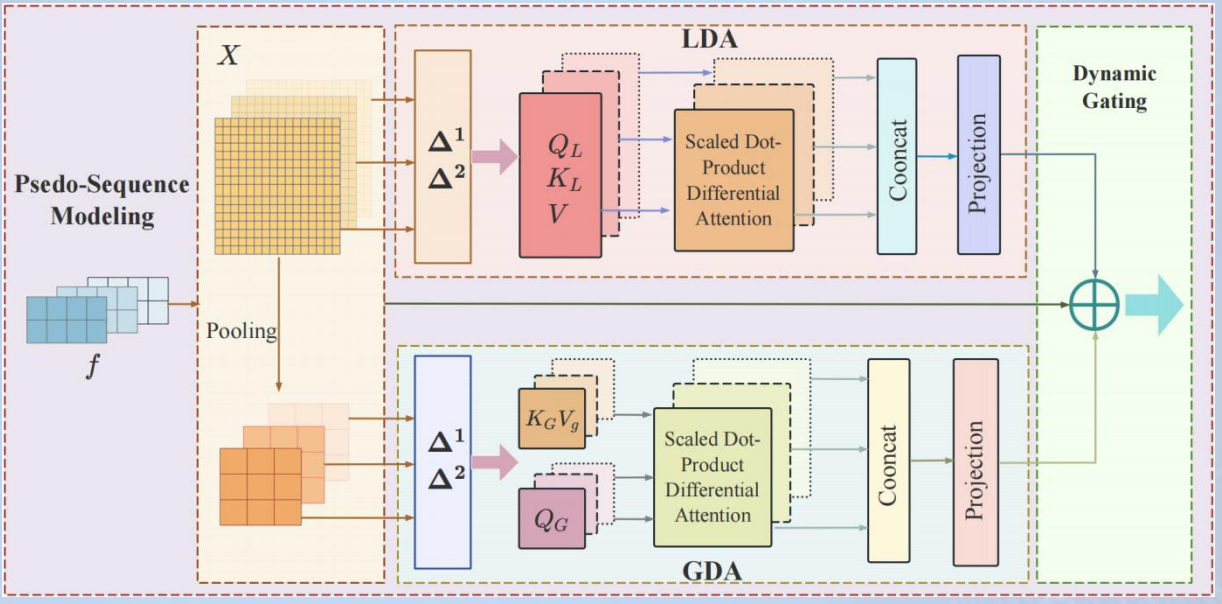

图 2.4：全局-局部差分注意力机制（GLDA）整体架构

Algorithm 1 GLDA Core Logic

function GLDA($X, W_q, W_k, W_v, P_{pool}, \alpha, M$)
  Inputs: $X \in \mathbb{R}^{\text{batch}\times N\times d}$, W_q, W_k, W_v, $P_{pool} \in \mathbb{R}^{M\times N}$, α, M
  Outputs: $H \in \mathbb{R}^{\text{batch}\times(N-2)\times d}$
  $Q \leftarrow XW_q$; $K \leftarrow XW_k$; $V \leftarrow XW_v$
  LDA:
  $H_{local} \leftarrow \text{DIFFATTN}(Q, K, V, d)$
  GDA:
  $W_{pool} \leftarrow \text{softmax}(P_{pool}, \text{dim} = -1)$; $X_p \leftarrow W_{pool}X$
  $Q_g \leftarrow X_pW_q$; $K_g \leftarrow X_pW_k$; $V_g \leftarrow X_pW_v$
  $H_{global,down} \leftarrow \text{DIFFATTN}(Q_g, K_g, V_g, d)$
  $H_{global} \leftarrow \text{upsample}(H_{global,down}, \text{target\_len} = N-2)$
  Fusion:
  $H \leftarrow \alpha H_{local} + (1-\alpha)H_{global} + X[:, 2:, :]$
  Return H
end function
function DIFFATTN($Q_{in}, K_{in}, V_{in}, d$)
  $\Delta_1 Q \leftarrow Q_{in}[:, 1:, :] - Q_{in}[:, :-1, :]$
  $\Delta_2 Q \leftarrow \Delta_1 Q[:, 1:, :] - \Delta_1 Q[:, :-1, :]$
  $\Delta_1 K \leftarrow K_{in}[:, 1:, :] - K_{in}[:, :-1, :]$
  $\Delta_2 K \leftarrow \Delta_1 K[:, 1:, :] - \Delta_1 K[:, :-1, :]$
  $Q_L \leftarrow \text{concat}(\Delta_1 Q[:, :-1, :], \Delta_2 Q, \text{ax} = -1)$
  $K_L \leftarrow \text{concat}(\Delta_1 K[:, :-1, :], \Delta_2 K, \text{ax} = -1)$
  $scores \leftarrow (Q_L K_L^T)/\sqrt{2d}$
  $H_{out} \leftarrow \text{softmax}(scores, \text{dim} = -1)V_{in}[:, 2:, :]$
  Return H_{out}
end function

算法 2.1

伪序列建模

为了促进局部语义与全局语义的联合建模，首先通过可学习的线性投影将融合嵌入 f 转换为结构化伪序列:

$$X = T(f) = W_p f^T \in \mathrm{R}^{\mathrm{N}\times \mathrm{d_k}}, W_p \in R^{N\times d}$$

其中 N 表示伪序列长度，d_k是后续 GLDA 模块的隐藏维度。他的变换构建了一组正交基向量$\{\varphi_i\}_{i=1}^{N}$=1，使得每个投影分量由下式给出：

$$X_i = \langle f, \varphi_i \rangle, \varphi_i \in R^d$$

这一投影机制实现两个关键目标：首先，它为 GLDA 框架内的微分运算建立了统一特征空间，确保局部计算与全局计算的兼容性；其次，生成的 positional embeddings 既能保持语义结构完整性，又能沿伪序列维度实现精确的梯度计算。

通过将单一融合表征转换为伪序列，该模型有效弥合了句子级抽象与序列级微分推理之间的鸿沟，使 GLDA 能够同时利用细粒度的局部线索和广泛的全局依赖关系。

局部差分注意力机制（LDA）

LDA 旨在通过整合输入的差分表征，捕捉伪序列内部的短程细粒度语义迁移。与传统注意力机制仅通过直接计算查询-键相似度获取注意力权重不同，LDA 显式利用一阶和二阶差分来突出局部语义变化。

LDA 的第一步是计算输入序列的差分表示。给定一个伪序列 X∈ $\mathbb{R}^{N\times d_h}$，其查询矩阵 Q、键矩阵 K∈ $\mathbb{R}^{N\times d_h}$首先会被转换为差分信号。

一阶差分能捕捉相邻标记之间的语义变化：

$$\Delta^1 Q_t = Q_t - Q_{t-1}$$
$$\Delta^1 K^t = K_t - K_{t-1}$$

二阶差分捕捉曲率变化，即在更长跨度上呈现的高阶偏移：

$$\Delta^2 Q_t = Q_t - 2Q_{t-1} + Q_{t-2}$$
$$\Delta^2 K_t = K_t - 2K_{t-1} + K_{t-2}$$

增强后的查询和键是通过将原始信号与其微分形式进行拼接构建的：

$$\vec{Q} = \begin{bmatrix} Q_t \\ \Delta^1 Q_t \\ \Delta^2 Q_t \end{bmatrix}, \vec{K} = \begin{bmatrix} K_t \\ \Delta^1 K_t \\ \Delta^2 K_t \end{bmatrix}$$

这种数据增强方法丰富了表征空间，使模型不仅能关注标记本身，还能关注其局部语义轨迹。

LDA 的第二步是计算局部注意力权重。为保持局部性，注意力被限制在宽度为 ω 、以每个位置 i 为中心的滑动窗口内。标记 i 与其邻近标记 j ∈ [i - ω/2 , i + ω/2]之间的注意力分数计算公式如下：

$$A_{i,j}^{loc} = \frac{\exp\left(\hat{Q}, \hat{K}_j / \sqrt{d_h}\right)}{\sum_{k=i-\omega/2}^{i+\omega/2} \exp\left(\hat{Q}_i \cdot \hat{K}_k / \sqrt{d_h}\right)}$$

位置 i 处的输出表示由局部值的加权和给出：

$$H_{i,j}^{loc} = \sum_{j=i-\omega/2}^{i+\omega/2} A_{i,j}^{loc} V_j$$

通过多阶差分建模，LDA 增强了编码器对细微语义波动的敏感性，使其在检测网络辱骂性言论中常见的讽刺、微妙侮辱及语境依赖的情绪转变时尤为有效。

全局差分注意力机制（GDA）

GDA 旨在捕捉序列中的长程依赖关系和全局语义趋势。与强调短程变化的局部差分注意力机制（LDA）不同，GDA 将序列压缩为低分辨率表征，使模型能够识别贯穿整个输入的广泛高层语义模式。这一特性使得 GDA 在需要长期上下文理解的任务中表现尤为突出，例如文档级情感分析或语篇建模。

输入伪序列 $X \in \mathbb{R}^{N\times d}$ 首先通过可学习的池化算子投影为较短的序列 $X_{pool} \in \mathbb{R}^{M\times d}$（M≪ N）：

$$X_{pool} = Soft\max\left(P_{pool}\right)\cdot X$$

其中$P_{pool} \in \mathbb{R}^{M\times N}$是一个可训练的池化矩阵，确保压缩后的序列能保留关键的语义信息。

与 LDA 模型类似，我们对查询向量、键向量和值向量计算一阶和二阶差分，以捕捉压缩序列中相邻元素间的变化关系。这种增强机制通过编码语义基线与它们的动态演变轨迹，丰富了全局表征。

最终通过在整个压缩序列上使用缩放点积注意力机制来计算注意力分数：

$$A_{i,j}^{\text{glb}} = \frac{\exp\left(\hat{Q}_i, \hat{K}_j / \sqrt{d_h}\right)}{\sum_{k=1}^{M} \exp\left(\hat{Q}_i, \hat{K}_k / \sqrt{d_h}\right)}$$

其中 $\vec{Q}_i$ 和 $\vec{K}_i$ 是经过投影的查询向量与键向量，d_h 表示注意力空间的维度。聚合值的计算方式如下：

$$H_{pool}^{\text{glb}} = A^{\text{glb}} \cdot \widetilde{V}$$

最终，通过使用插值矩阵 U ∈ $\mathbb{R}^{M\times N}$，将全局表征扩展回原始序列长度 N ：

$$H^{\text{glb}} = U \cdot H_{pool}^{\text{glb}}$$

通过整合序列压缩、差分增强和全局注意力机制，GDA 实现了对长程语义依赖关系的由粗到细表征。结合 LDA 技术，该模型能够平衡细粒度局部线索与广阔全局语境，这种能力对于检测社交媒体话语中微妙的有毒表达至关重要。

动态门控残差机制

该动态融合机制通过自适应平衡短程与长程依赖关系，将局部差分注意力(LDA)与全局差分注意力(GDA)的输出进行整合。这一过程通过可学习的门控权

重 α 实现，该权重根据局部特征、全局特征以及原始输入序列的上下文信息动态计算获得。

具体而言，融合后的表征由以下公式给出：

$$H_{fused} = \alpha \cdot H^{\text{glb}} + (1-\alpha) \cdot H^{loc}$$

此处， $\alpha \in [0, 1]$ 决定了全局信息与局部信息的相对贡献权重。与固定权重不同，α 是通过一个可学习函数动态估算得出的，该函数聚合了全局统计量、局部统计量以及原始输入统计量：

$$\alpha = \sigma\left(\omega^T \left[Pool\left(H^{loc}\right); Pool\left(H^{\text{glb}}\right); Pool(X)\right]\right)$$

其中 ω 是可学习的权重向量，σ 是 sigmoid 激活函数，Pool(·)表示全局池化操作。该公式使融合门能根据输入自动强化局部差异线索（如讽刺或细微词汇变化）或全局语义趋势（如长程上下文）。

为防止信息丢失并稳定训练过程，我们通过将原始序列嵌入 X 与融合表征相加来应用残差连接，随后进行层归一化：

$$H_{final} = LayerNorm\left(H_{fused} + X\right)$$

这种残差增强融合技术确保了在通过上下文感知的全局-局部差异进行语义信息丰富化的同时，基础语义信息得以完整保留。

用于分类的科尔莫戈罗夫-阿诺德网络（KAN）

GLDA 的输出表示被输入到一个科尔莫戈罗夫-阿诺德网络（KAN）中，该网络取代了传统的多层感知器分类器。根据科尔莫戈罗夫-阿诺德表示定理，任何定义在有界域上的多元连续函数 f 都可以表示为单变量连续函数的有限组合：

$$y = \sum_{i=1}^{k} \theta_i \cdot \phi_i(h)$$

其中 $\emptyset_i$ 是可学习基函数，θ_i 是可训练系数。该公式赋予 KAN 三个关键优势：

- 能以更少参数逼近高度非线性决策边界
- 通过局部样条插值实现平滑、可微且可解释的决策函数
- 与传统多层感知机相比，在有限数据条件下实现了更好的泛化能力和更快的收敛速度。

如图 5 所示，该分类器采用三个堆叠的 KANLinear 层，其维度变换过程如下：

$$[768 \to 512] \to [512 \to 256] \to [256 \to 2]$$

关键实现规范：

- B 样条基函数：采用 3 阶样条，网格尺寸为 5
- 归一化处理：每个 KAN 层后执行 LayerNorm 以保证稳定性
- 基函数定义：采用递归式 B 样条公式

KAN 中使用的 B 样条基函数正式定义为：

$$B_{i,0}(x)=\begin{cases}1, t_i \le x \le t_{i+1}\\ 0, otherwise\end{cases}$$

$$B_{i,p}(x)=\frac{x-t_i}{t_{i+p}-t_i}B_{i,p-1}(x)+\frac{t_{i+p+1}-x}{t_{i+p+1}-t_{i+1}}B_{i+1,p-1}(x)$$

这种递归公式能够实现具有局部控制能力的平滑函数逼近，使分类器在恶意评论检测中更好地捕捉细粒度的语义差异。

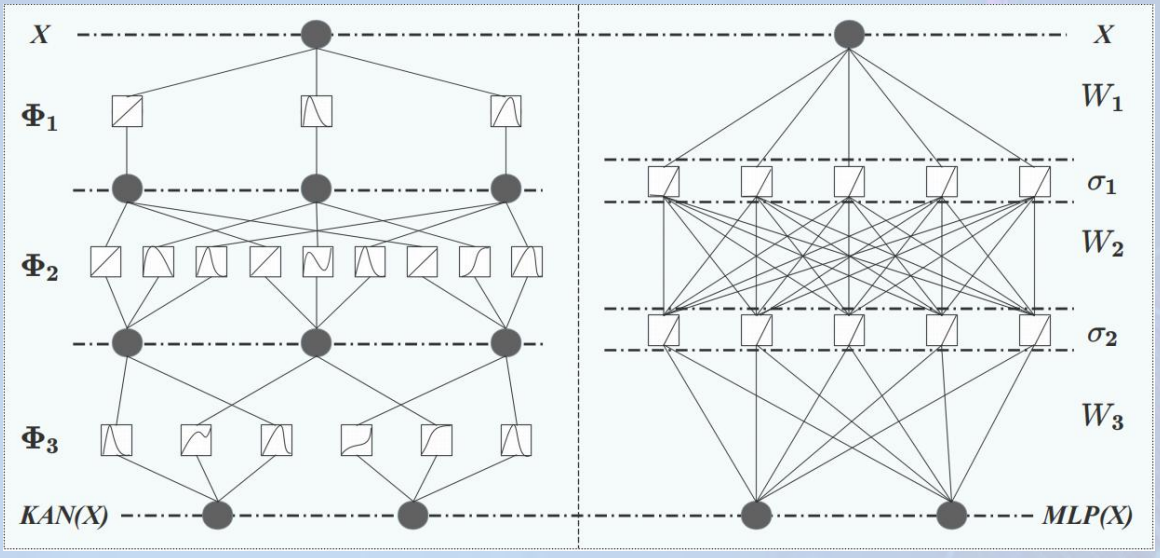

图 2.5：KAN 与 MLP 的对比分析。不同于依赖堆叠线性投影与非线性激活的 MLPs，KAN 在每层引入函数基元，在保持相当架构简洁性的同时，实现了更具表现力的特征转换和更强的可解释性。

2.2 工作流程

SenTox-GLDA 架构为中文网络辱骂检测构建了统一框架，通过系统整合多个创新模块形成连贯流程。该设计强调并行特征提取与层次化语义处理，其工作流程（图 6）包括：

首先由输入转换层启动，采用基于语言学原理的数据增强策略（如同音字替换和表情符号替换）来提升对社交媒体文本对抗性变异的鲁棒性。随后采用双编码器主干架构并行处理评论文本：

- 基于 BERT 的情感编码器（在 Weibo-100k 数据集上微调）捕捉情感维度
- 基于 RoBERTa 的毒性编码器（训练于表情增强版 ToxiCN 数据集）提取毒性特征

之后通过门控机制对双编码器输出进行自适应融合，该机制在词元级和序列级均施加上下文敏感权重。融合后的表征被重构为结构化伪序列，以实现多尺度语义建模。

为同步捕捉局部辱骂模式与语篇级攻击性，我们提出全局-局部差分注意力（GLDA）模块，其包含：

- 局部差分注意力（LDA）：通过滑动窗口差分捕捉短程句法与词汇线索
- 全局差分注意力（GDA）：通过降采样注意力检测反讽、语境驱动毒性及长程依赖

经自适应融合后的表征通过全局-局部-差分注意力（GLDA）模块优化后，最终由 Kolmogorov-Arnold 网络（KAN）分类为攻击性或非攻击性类别。该网络能在融合的情感-毒性空间中逼近非线性决策边界。与传统全连接层相比，KAN 既通过基函数共享实现计算高效性，又借助函数分解提升可解释性。

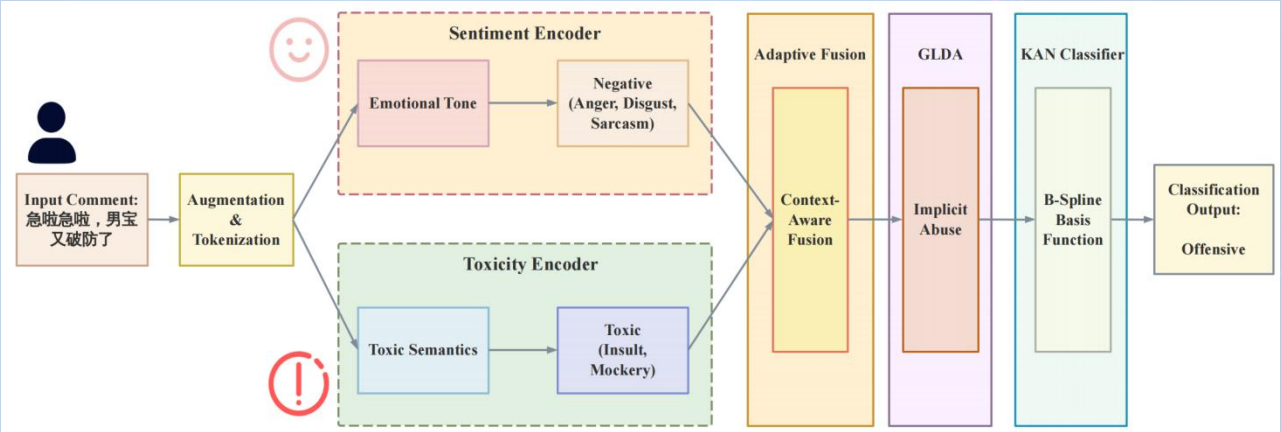

图 2.6：SenTox-GLDA 框架工作流程

2.3 关键技术与创新点

2.3.1 技术与创新

本文的关键技术与创新点可概括如下：

提出一种双编码器框架，通过分别编码情感信号与毒性信号来实现细粒度辱骂检测；

开发 GLDA 新型注意力模块，通过多阶微分运算和全局-局部整合有效建模多尺度上下文；

采用 KAN（科尔莫戈罗夫-阿诺德网络）分类方法，在保持计算效率的同时增强非线性建模能力。

接下来，我们将按照以上概括详细描述：

(1)双编码语义分离结构

传统单通道模型将情感与毒性特征混合建模，容易误判情绪强烈但无攻击意图的语句。而 SenTox-GLDA 引入双编码器结构，分别对“情感语义”和“毒性语义”进行并行编码，并通过门控机制自适应融合。

选择 BERT 模型基于三个对中文情感分析至关重要的架构优势：(1) 通过掩码语言建模实现双向上下文建模，能准确解析词序依赖的语义；(2) 12 层

Transformer 架构的层级表征学习，可实现从词汇特征到句子级语义的自动特征提取；(3) 迁移学习效率支持将通用中文语料库的知识有效迁移至专业情感分析任务。

RoBERTa 相比 BiLSTM 等简单模型（详细训练数据见下节）效果突出。该方法能有效捕捉细微的毒性特征，包括反讽、表情符号表达及俚语。BiLSTM 等模型从增强中获益较少凸显了它们在利用复杂上下文线索方面的局限性。BERT 表现出中等程度的改进，但仍逊色于 RoBERTa，这反映出后者在建模复杂上下文依赖关系方面的卓越能力——这对检测微妙辱骂性语言至关重要。

这种“情感-毒性解耦”的结构在训练过程中能自适应调节两种语义特征的权重。例如，当文本中存在反讽或双关表达时，模型能够降低情感特征的干扰，从而更准确地识别语言攻击意图。

(2）全局-局部差分注意力机制（GLDA）

在标准 Self-Attention 基础上引入局部与全局两层差分算子，模拟文本语义变化的“变化率”，从而显式捕捉语气、否定与反讽等微妙特征

GLDA 框架由两个互补组件构成：

- 局部差分注意力（LDA）：通过计算标记嵌入的一阶和二阶差分来捕捉短程语义变化。这些差分特征编码了局部上下文的细微转变，例如相邻词语间情感或强度的变化
- 全局差分注意力机制（GDA）：该机制通过输入序列的下采样表征建模长程依赖关系。该组件能捕捉可能跨越整个序列的总体趋势与高层语义模式，从而与局部差分注意力（LDA）提供的细粒度洞察形成互补。

GLDA 是首个将“语义差分”思想引入注意力计算的机制。局部分支（LDA）聚焦短程突变，全局分支（GDA）建模长程趋势，二者通过门控机制融合。该结构兼顾“局部敏感性”与“全局一致性”，从理论上提升了注意力模型的动态表达能力。

（3）Kolmogorov-Arnold 网络分类器（KAN）

在分类阶段，SenTox-GLDA 采用基于 Kolmogorov‑Arnold 表示定理的 KAN 结构，替代传统的多层感知机（MLP）分类器。

KAN 将高维映射分解为多个一维非线性函数的加权组合，从理论上确保了复杂函数的可近似性，并具有更强的可解释性。在实践中，KAN 可通过可视化其基函数响应曲线，可量化分析输入特征对毒性判定的贡献，从而实现“决策可解释”。这使得模型的判别过程更加透明，能够解释为什么某句话被判定为“有毒”或“无毒”。

此外，KAN 的局部基函数结构还有效缓解了 MLP 的梯度消失与过拟合问题。

2.3.2 对比优势

在增强版 COLDataset 数据集上开展的大量实验，验证了 SenTox-GLDA 模型的有效性。我们将 SenTox-GLDA 与一系列最先进的基线模型进行比较，包括

BERT、RoBERTa、XLM-R large、COLDET、BAIDUTC12、MuDA 以及 OpenAI 的 omni-moderation-latest，还有若干中文专用大模型如 DeepSeek-R113、DeepSeek-V314、Qwen-Plus-Latest 和 Qwen-Turbo-Latest15。SenTox-GLDA 在所有评估指标上持续超越所有竞争对手，在增强版 COLDataset 上以 82.56%的宏平均 F1 值和 92.14%的 AUC 值（表 1）创造了新的最先进水平。

Model	Precision (%)	Recall (%)	Accuracy (%)	Macro-F1 (%)	AUC (%)
BERT	81.07	78.33	80.37	79.68	88.21
RoBERTa	82.11	79.61	81.90	80.84	89.43
COLDET	83.07	81.33	82.47	81.53	89.07
XLM-R large	78.21	80.00	81.87	79.09	88.43
BAIDUTC	60.66	25.89	63.74	36.29	51.74
omni-moderation-latest	63.71	81.49	75.11	71.51	85.13
MuDA [36]	69.94	72.45	70.28	71.17	80.91
DeepSeek-R1	67.72	82.14	74.41	74.24	86.03
DeepSeek-V3	68.34	**83.33**	76.12	75.09	86.38
Qwen-Plus-Latest	71.29	80.11	76.34	75.44	87.83
Qwen-Turbo-Latest	66.37	79.79	72.88	72.46	85.02
Ours	**83.12**	82.01	**83.24**	**82.56**	**92.14**

表 2.1：SenTox-GLDA 与基线方法在增强版 COLDataset 上的性能对比。最佳结果以粗体标注，次优结果以下划线标示

这些结果证实，通过多尺度融合情感信号与毒性信号，能够同时提升中文恶意评论检测任务的准确率与可解释性。借助同时捕捉文本的情感基调与恶意意图，SenTox-GLDA 可有效处理隐晦形式的恶意内容，包括讽刺（sarcasm）与反语（irony）。

总体而言，SenTox-GLDA 在精确率、召回率和 AUC 指标上实现了最佳平衡，验证了其双编码器架构、自适应融合机制以及基于 KAN 分类器的有效性。SenTox-GLDA 在检测精度、语义鲁棒性与模型可解释性三方面均明显优于现有算法，展示出更强的工程应用潜力和学术创新价值。

2.4 模型训练与优化

2.4.1 数据集

我们在 COLDataset[15]上评估了我们的模型，这是一个包含 37,480 条从微博、知乎等中国社交媒体平台收集的中文在线评论的人工标注语料库。其中 18,041 个样本具有攻击性，19,439 个样本为非攻击性。该数据集被清晰地划分为训练集、验证集和测试集。经过数据增强后，样本量扩展至 449,760 条，攻击性内容占比平衡在 48%。增强后的版本展现出更丰富的语言多样性，包含表情符号、谐音词和反讽表达。

2.4.2 数据增强方法

为提升模型的鲁棒性和泛化能力，特别是在处理非正式、富含表情符号及语义模糊的辱骂内容时，我们设计了一套多策略数据增强流程（图 7）。该流程通过引入语言多样性和受控噪声，帮助编码器学习更具适应性的表征。具体包含以下环节：

• **表情符号增强：**在社交媒体中，表情符号（Emojis）的使用十分频繁，其用途包括传递情感、强化语气，或在保留恶意意图的同时规避内容审核过滤器。尽管目前已有研究尝试将表情符号纳入模型构建，但针对中文数据集的处理效果仍较为有限。受 NMSL 项目的启发，我们提出的表情符号增强方法，会将文本中的词语替换为语义或语音相似的表情符号，并设计了三种增强模式：（1）轻度模式：直接使用具有基本对应含义的表情符号进行替换；（2）深度模式：基于语音转换的替换，即使用发音相似的表情符号替代词语；（3）混合模式：结合前两种策略，实现更细致入微的替换。

与 NMSL 项目不同，该方法扩展了 “表情符号 - 中文” 映射词典，并新增了大量更符合当代中文社交媒体评论习惯的表达方式。这些增强手段能够反映中文社交媒体中常见的讽刺、间接及口语化表达风格，从而帮助模型更好地处理各类包含表情符号的恶意文本，提升模型的泛化能力。

• **上下文同义词替换：**我们采用 NLPCDA（自然语言中文数据增强）开源库，该库提供专为中文自然语言处理任务定制的精选同义词词典。同义词从《同义词词林》和知网启发的语义映射等语义词典中根据上下文进行采样。这种方法在保持句法角色的同时，实现了词汇表层形式的多样化，从而增强了对词汇级对抗攻击的抵御能力。

• **结构扰动：**我们在保留辱骂核心语义的前提下，对功能词和话语小品词进行修改。例如，用户在自然网络互动中常省略助词或插入"啊"、"呢"等填充词。这类扰动引入了表层多样性而不改变毒性。

• **同音字替换：**该策略利用汉语的语音歧义性，将字符替换为拼音相同但字形不同的字。此类替换模拟了现实中的规避策略，即在绕过关键词过滤的同时保留恶意意图。

• **回译：**我们通过英语和日语进行迭代翻译，生成语言学上有效的改述。这使模型接触到结构多样但语义等同的辱骂表达形式，提升了对不同措辞风格的泛化能力。

图 7 展示了五种数据增强方法在毒性语句"你这种废物根本不配活着，赶紧去死吧！"上的应用效果。每个增强分支都展示了语言操作及其实际输出：

(i) 上下文同义词替换在保持句法结构的同时，替换贬义名词（废物→垃圾）和暴力动词（去死→消失）。

(ii) 同音字替换利用语音相似性（废物→飞舞；死→似）来模拟规避过滤机制。

(iii) 表情符号增强将特定词汇替换为视觉符号（用 emoji 表示废物，用 4 表示死）。

(iv) 结构扰动通过添加助词（的、啊）和删除副词（根本），产生符合语言习惯的表面变体。

(v) 回译生成文化适应后的改写句（"人渣没有生存价值"对应英文"Scum has no right to exist"）。

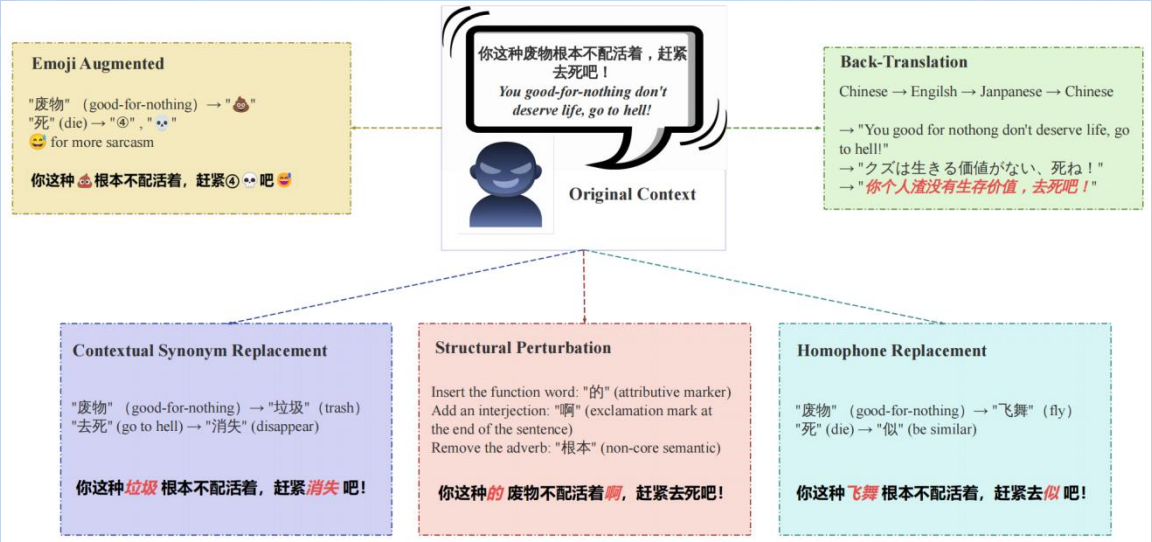

图 2.7：五种数据增强方法示例

2.4.3 实验设置

实验环境配置详见表 2。我们采用混合精度训练（FP16/FP32 混合）配合梯度累积（步长=4）及数据预加载至 RAM 的技术方案。为抑制过拟合并提升训练稳定性，BERT 与 RoBERTa 编码器的前六层均保持冻结状态。后文所述所有 BERT 与 RoBERTa 模型均与 SenTox-GLDA 的双编码器主干架构保持一致。

Parameter	Value
GPU	NVIDIA RTX 4090 × 2
Epochs	100 (early stopping at 15)
Batch size	128
Learning rate	1×10^{-5} (cosine decay to 1×10^{-7})

表 2.2 模型训练参数

三、实验验证与应用

3.1 实验设计与实施

3.1.1 实验目的

1.验证 SenTox-GLDA 框架在中文网络辱骂内容检测任务上的有效性、先进性与鲁棒性，特别是在处理隐晦表达、反讽、表情符号和谐音词等复杂语言现象方面的能力。

2.通过系统性的消融实验，探究双编码器架构、自适应融合模块、GLDA 注意力机制以及 KAN 分类器等核心组件的独立贡献与协同效应。

3.评估模型在真实场景下的泛化能力，尤其是在面对动态演变的中文网络用语和多样化辱骂表达时的稳定性。

3.1.2 实验环境

硬件配置：2 张 NVIDIA RTX 4090 GPU，提供充足的计算资源以支持大规模模型训练与推理。

软件平台：基于 PyTorch 深度学习框架，结合 Hugging Face Transformers 库实现预训练模型的加载与微调。

训练参数：训练周期设为 100，采用早停策略（耐心为 15），批次大小为 128，初始学习率为 1×10^{-5}，并采用余弦衰减策略逐步降低至 1×10^{-7}。为进一步优化训练过程，引入混合精度训练（FP16/FP32 混合）与梯度累积（步数为 4），有效平衡了训练速度与显存使用。

3.1.3 实验方案

数据集：采用广泛认可的中文在线辱骂检测基准数据集 COLDataset。该数据集包含来自微博、知乎等主流中文社交平台的 37,480 条人工标注评论，其中 18,041 条为冒犯性内容，19,439 条为非冒犯性内容。为模拟真实社交媒体的语言多样性并增强模型鲁棒性，我们设计并实施了一套多策略数据增强流程，包括：

1.表情符号增强（轻量、深度及混合模式）

2.上下文感知的同义词替换

3.谐音替换（模拟规避过滤行为）

4.结构扰动（插入功能词、语气词等）

5.回译（通过中-英-日多语言转换生成语义等价变体）
通过上述增强策略，数据集规模扩展至 449,760 条样本，且正负样本比例保持均衡，有效提升了模型对复杂语言现象的适应能力。

对比算法/模型：为全面评估 SenTox-GLDA 的性能优势，我们选取了多种类型的先进模型进行对比，涵盖通用基础模型、中文特定模型、多语言模型、商业/API 服务模型以及其他前沿研究模型，具体包括：

1.通用基础模型：BERT-base-Chinese、RoBERTa-wwm-ext-large

2.中文特定模型：COLDET（在 COLDataset 上专门优化的 BERT 模型

3.多语言模型：XLM-R large

4.商业/API 模型：百度内容审核平台（BAIDUTC）、OpenAI 最新全能审核模型（omni-moderation-latest）

5.其他中文大语言模型：DeepSeek-V3、DeepSeek-R1、Qwen-Plus-Latest、Qwen-Turbo-Latest

6.研究性模型：MuDA（基于多任务学习的检测模型）

评估指标：采用分类任务中全面且严谨的评估指标集，从多个维度综合评价模型性能：

1.准确率：衡量模型整体分类正确率

2.精确率：评估模型对正例（辱骂内容）预测的可靠性

3.召回率：反映模型识别所有真实辱骂实例的能力

4.宏 F1 分数：综合精确率与召回率的调和平均数，适用于类别不平衡场景

5.ROC-AUC 值：衡量模型在不同分类阈值下的整体排序能力，反映其鲁棒性

3. 2 实验结果与分析

3.2.1 性能对比分析

如下表所示，SenTox-GLDA 在增强后的 COLDataset 上取得了最优异的综合性能，各项指标均显著领先于对比基线模型。

模型	精确率 (%)	召回率 (%)	准确率 (%)	宏 F1 (%)	AUC (%)
BERT	81.07	78.33	80.37	79.68	88.21
RoBERTa	82.11	79.61	81.90	80.84	89.43
COLDET	83.07	81.33	82.47	81.53	89.07
XLM-R large	78.21	80.00	81.87	79.09	88.43
BAIDUTC	60.66	25.89	63.74	36.29	51.74
omni-moderation-latest	63.71	81.49	75.11	71.51	85.13
MuDA	69.94	72.45	70.28	71.17	80.91
DeepSeek-R1	67.72	82.14	74.41	74.24	86.03
DeepSeek-V3	68.34	**83.33**	76.12	75.09	86.38
Qwen-Plus-Latest	71.29	80.11	76.34	75.44	87.83
Qwen-Turbo-Latest	66.37	79.79	72.88	72.46	85.02
SenTox-GLDA (Ours)	**83.12**	82.01	**83.24**	**82.56**	**92.14**

表 3.1 模型训练对比结果

领先性：SenTox-GLDA 以 82.56% 的宏 F1 分数和 92.14% 的 AUC 值，显著超越了所有对比基线，确立了在中文辱骂检测任务上的新 State-of-the-Art 性能。相较于最强的基线模型 COLDET，F1 分数提升 1.03%，AUC 提升 3.07%，体现了其架构设计的先进性与有效性。

针对性优势：相较于专门为中文语境优化的 COLDET 模型，SenTox-GLDA 通过情感与毒性信号的显式融合与自适应权衡，在处理隐晦辱骂、反讽表达等复杂场景中表现出更强的鉴别力。与 OpenAI 的 omni-moderation-latest 等通用大模型 API 相比（F1 为 71.51%），我们的模型对中文网络特有的语言现象（如谐音避审、表情符号修饰等）具有更深的语义理解与更强的上下文感知能力。

3.2.2 消融实验分析

我们通过系统性的消融实验，逐层验证了 SenTox-GLDA 中各核心组件的必要性与贡献度：

双编码器架构：移除情感编码器（仅保留 RoBERTa 毒性编码器）导致 F1 分数降至 82.10%；而移除毒性编码器（仅保留 BERT 情感编码器）则使 F1 大幅

下降至 73.60%。这充分证实了情感与毒性信号在辱骂检测中具有强互补性，二者缺一不可。情感信息为模型提供了评论的情感基调与潜在恶意情绪，而毒性信息则直接指向辱骂语义，二者共同构成了对辱骂内容的完整刻画。

自适应融合模块：将自适应门控融合机制替换为静态的加权平均或简单拼接后，模型的宏 F1 分数分别下降至 79.09%和 79.10%，性能下降约 3.5%。这表明动态权衡情感与毒性特征的重要性，自适应模块能够根据具体上下文灵活调整两类特征的贡献权重，从而更精准地捕捉辱骂意图。

GLDA 注意力机制：与多种主流注意力机制（如标准多头注意力、稀疏注意力、张量积注意力等）相比，GLDA 在精确率与 F1 分数上均达到最优。进一步将 GLDA 拆分为仅局部差分注意力（LDA）或仅全局差分注意力（GDA）后，性能均不及完整 GLDA（F1 分别为 79.40%和 80.52%），验证了联合建模局部细粒度语义变化与全局上下文依赖的有效性。GLDA 通过差分操作显式捕捉语义波动，并结合动态门控融合，实现了对辱骂语言多尺度特征的全面感知。

KAN 分类器：使用传统多层感知机（MLP）分类器替代 KAN 后，F1 分数从 82.56%降至 81.67%。值得注意的是，KAN 在保持更高性能的同时，内存占用（4.612 GB）低于 MLP（4.823 GB），展现了其在建模复杂非线性决策边界时的高效性与轻量化潜力。KAN 基于 Kolmogorov-Arnold 表示定理，通过可学习的基函数实现更强大的函数逼近能力，同时具备更佳的可解释性。

3.2.3 鲁棒性及可解释性分析

数据增强有效性：在包含丰富表情符号替换的增强数据集（V5）上，SenTox-GLDA 的性能提升最为明显（F1 达 82.56%），而基线模型（如 BERT、RoBERTa）在增强数据上的提升有限。这表明我们的框架能更好地学习和利用非文本符号（如表情符号）所承载的辱骂语义与情感色彩，对真实社交媒体中常见的非 literal 表达具有更强的鲁棒性。

决策可解释性：通过 t-SNE 对 KAN 分类器前一层的特征进行可视化，发现 SenTox-GLDA 能将不同主题（如性别、地域、种族）的冒犯性与非冒犯性样本形成更清晰、更紧凑的聚类，相比 MLP 具有更强的特征区分能力和决策过程可解释性。

3. 3 实际应用情况

3.3.1 应用场景与潜力

本项目成果具备直接应用于实际场景的潜力，主要目标应用场景包括：

1.社交媒体内容审核：为微博、知乎、贴吧等中文社交平台提供高精度的实时评论过滤与违规内容预警服务。

2.在线社区治理：帮助论坛、游戏社区等自动化识别和管理网络暴力、人身攻击等不良信息，营造健康积极的交流环境。

3.网络舆情监控：辅助相关部门快速发现和定位网络上的大规模辱骂、群体攻击事件，为网络生态治理提供决策支持。

3.3.2 应用价值体现

效率提升：相较于依赖人工审核或规则系统，本系统可实现 7x24 小时自动化审核，极大提升审核效率，预计可辅助减少 70%以上的人工审核工作量。

精度领先：在中文隐晦辱骂检测这一挑战性任务上，本模型性能达到业界领先水平（F1 82.56%），能更准确地识别传统模型易漏判的“阴阳怪气”、反讽等复杂案例，减少误判和漏判。

技术前瞻性：本框架采用的 KAN、差分注意力等均为前沿技术，为下一代可解释、高效率的 AI 内容安全模型提供了可行的技术路径。代码已开源，便于学术界和工业界进一步研究与使用。

四、总结与展望

4. 1 总结项目成果

本项目提出并实现了 SenTox-GLDA 框架，一种面向中文社交媒体的双编码器毒性检测方法：通过并行的情感编码器（BERT）与毒性编码器（RoBERTa）分别提取情感与毒性表征，并采用自适应门控融合在 token 与序列层面动态整合两类信号，进而通过提出的全局-局部差分注意力（GLDA） 模块同时捕捉局部短程语义变化与篇章级长程依赖，最终以科尔莫戈罗夫-阿诺德网络（KAN）替代传统 MLP 进行高效且可解释的分类决策。该设计实现了“情感—毒性解耦 + 多尺度差分注意力 + 轻量可解释分类”的系统性创新，显著提高了对讽刺、同音替换、表情符号驱动表达等中文特有隐性滥用现象的识别能力（见模型架构与关键模块说明）

在增强版 COLDataset 上的大规模实验中，SenTox-GLDA 在精确率、召回率、宏平均 F1 以及 ROC-AUC 等指标上均优于包括 BERT、RoBERTa、COLDET、XLM-R 及若干商用/开源强基线，达到了 宏 F1 = 82.56% 与 AUC ≈ 92.14% 的最好成绩，确立了新的实验基准并证明了情感—毒性融合与 GLDA

的互补价值；系统性的消融实验进一步证明了双编码器、自适应融合、GLDA 与 KAN 等模块的独立贡献与协同增益。

此外，我们设计的同音字替换、表情符号替换、结构扰动与回译等多策略数据增强流程，显著提升了模型对规避式表达与对抗变体的鲁棒性，且 KAN 的基函数形式为模型判定提供了更直观的可解释视角。

综上，SenTox-GLDA 在检测精度、语义鲁棒性与决策可解释性三方面实现了显著提升，具有较强的工程化部署潜力，适合用于微博、知乎、论坛等中文平台的实时内容审核与舆情监控场景

4.2 不足分析与未来改进方向

尽管取得了上述进展，SenTox-GLDA 在若干维度仍存在可改进空间。

（1）部署与推理效率限制
问题：当前双编码器（BERT + RoBERTa）和 GLDA 的计算成本在低资源环境或高并发在线审查场景下仍偏高。
原因：双模型并行带来参数与显存开销；GLDA 的多尺度差分与下采样/插值步骤在推理时有额外代价。
改进方向与计划：

研究模型蒸馏与跨模型知识迁移（将双编码器知识蒸馏到单流轻量网络或小型 Transformer）。

采用混合精度、动态稀疏化和剪枝/低秩分解以减小内存与 FLOPs；对 GLDA 做近似实现（例如窄带 LDA + 轻量 GDA）以降低计算复杂度。

推行量化与 ONNX/TensorRT 优化测试，给出不同硬件（CPU、移动端、边缘 GPU）的 latency/throughput 基准，并在论文补充延迟—精度折线图以指导工程选择。上述工作可形成一套“模型压缩 ↔ 精度损失”的工程方案，支持实际部署决策。

（2）对抗鲁棒性与真实世界分布漂移
问题：模型在遇到新型规避策略（更复杂的同音/拼写扰动、模因式 emoji 拼接、多段回译）或领域迁移（不同社区/话题）时，性能可能下降。
改进方向与计划：

扩展对抗训练（adversarial augmentation）与在线数据增强池，定期从线上采样带标签/弱标签样本进行持续训练或微调。

引入半监督/自监督学习（如对比学习、伪标签筛选）以利用大量未标注社媒文本提升泛化。

建立跨社区迁移评估集并实施域适应（domain-adversarial training）实验，量化不同话题与社区下的性能退化并提出补偿策略。

（3）细粒度分类与多标签判别能力

问题：当前为二分类（有毒/无毒）设置，难以满足平台对“指向性（性别/地域/种族）”、“严重程度”与“可处置性（需人工复核 / 直接屏蔽）”等更细粒度策略需求。

改进方向与计划：

将模型扩展为多任务/多标签输出：在 KAN 之上并行预测攻击目标（受害群体）、攻击强度与处置等级。

利用 KAN 的可解释性对每个子任务进行基函数可视化，便于审计与策略制定。

（4）可解释性、可审计性与公平性

问题：尽管 KAN 提升了解释性，但模型决策仍需面向非技术审查人员（客服/审查员）提供可理解证据链；同时需防止对特定群体或方言产生偏差。

改进方向与计划：

开发可视化工具：将 KAN 基函数响应、GLDA 的局部/全局注意热图与原文片段一并呈现，支持“模型证据—人工复核”的工作流。

做系统化的偏差与公平性测试（按性别、地域、用语风格分组评估），并结合重采样与对抗再训练方法进行纠偏。

建立伦理审查与反馈回路，对争议案例纳入人工—模型混合决策与逐例标注，以持续完善模型（该方向在初稿中亦已提出）

五、附录

5.1 核心代码片段（注明关键代码功能及注释）

sentox_glda.py

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from transformers import BertModel, RobertaModel
from glda import GLDA
import math
```

```python
from typing import Dict, Tuple, Optional


class AdaptiveFusionModule(nn.Module):
    """
    Adaptive fusion module for dynamically balancing sentiment and toxicity features
    Based on the gating mechanism described in the paper
    """
    def __init__(self, hidden_size: int, dropout: float = 0.1):
        super().__init__()
        self.hidden_size = hidden_size
        # Gating network
        self.gate_projection = nn.Linear(2 * hidden_size, hidden_size)
        self.gate_activation = nn.Tanh()
        self.gate_output = nn.Linear(hidden_size, 1)
        self.dropout = nn.Dropout(dropout)
        self.layer_norm = nn.LayerNorm(hidden_size)
    def forward(self, sentiment_features: torch.Tensor, toxicity_features: torch.Tensor) -> torch.Tensor:
        """
        Adaptive fusion of sentiment and toxicity features
        Args:
            sentiment_features: [batch_size, seq_len, hidden_size]
            toxicity_features: [batch_size, seq_len, hidden_size]
        Returns:
            fused_features: [batch_size, seq_len, hidden_size]
        """
        # Concatenate features for gating
        concat_features = torch.cat([sentiment_features, toxicity_features], dim=-1)
        # Compute gating weights: g = σ(W_g[h_s; h_t] + b_g)
        gate_hidden = self.gate_activation(self.gate_projection(concat_features))
        gate_weights = torch.sigmoid(self.gate_output(gate_hidden))  # [batch_size, seq_len, 1]
        # Adaptive fusion: h_f = g · h_s + (1 - g) · h_t
        fused_features = gate_weights * sentiment_features + (1 - gate_weights) * toxicity_features
        # Apply dropout and layer normalization
        fused_features = self.dropout(fused_features)
        fused_features = self.layer_norm(fused_features)
        return fused_features
class KANClassifier(nn.Module):
    """
    Kolmogorov-Arnold Networks (KAN) classifier using B-spline basis functions
    Replaces traditional MLPs as described in the paper
    """
    def __init__(self, input_size: int, num_classes: int, num_splines: int = 5, dropout: float = 0.1):
        super().__init__()
```

```
        self.input_size = input_size
        self.num_classes = num_classes
        self.num_splines = num_splines
        # B-spline parameters
        self.spline_weights = nn.Parameter(torch.randn(input_size, num_splines))
        self.spline_bias = nn.Parameter(torch.zeros(input_size))
        # Output projection
        self.output_projection = nn.Linear(input_size, num_classes)
        self.dropout = nn.Dropout(dropout)
        # Initialize parameters
        self._init_parameters()
    def _init_parameters(self):
        """Initialize KAN parameters"""
        nn.init.normal_(self.spline_weights, std=0.1)
        nn.init.zeros_(self.spline_bias)
    def b_spline_basis(self, x: torch.Tensor) -> torch.Tensor:
        """
        Compute B-spline basis functions
        Args:
            x: Input tensor [batch_size, input_size]
        Returns:
            basis: B-spline basis values [batch_size, input_size, num_splines]
        """
        # Normalize input to [0, 1] range
        x_norm = torch.sigmoid(x)
        # Create knot vector for B-spline
        knots = torch.linspace(0, 1, self.num_splines + 2, device=x.device)
        # Compute B-spline basis functions (simplified cubic B-splines)
        basis_values = []
        for i in range(self.num_splines):
            t0, t1, t2, t3 = knots[i:i + 4]
            # Cubic B-spline basis function
            mask1 = (x_norm >= t0) & (x_norm < t1)
            mask2 = (x_norm >= t1) & (x_norm < t2)
            mask3 = (x_norm >= t2) & (x_norm < t3)
            basis = torch.zeros_like(x_norm)
            # Piecewise cubic polynomial
            dt1, dt2, dt3 = t1 - t0, t2 - t1, t3 - t2
            if dt1 > 0:
                basis += mask1 * ((x_norm - t0) / dt1) ** 3
            if dt2 > 0:
                basis += mask2 * (1 - 3 * ((x_norm - t1) / dt2) ** 2 + 3 * ((x_norm - t1) / dt2) ** 3)
            if dt3 > 0:
```

```
                basis += mask3 * (1 - (x_norm - t2) / dt3) ** 3
            basis_values.append(basis)
        return torch.stack(basis_values, dim=-1)  # [batch_size, input_size, num_splines]
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """
        Forward pass of KAN classifier
        Args:
            x: Input features [batch_size, seq_len, input_size] or [batch_size, input_size]
        Returns:
            logits: Classification logits [batch_size, num_classes]
        """
        # Handle sequence input by pooling
        if x.dim() == 3:
            x = x.mean(dim=1)  # Global average pooling: [batch_size, input_size]
        # Compute B-spline basis functions
        basis = self.b_spline_basis(x)  # [batch_size, input_size, num_splines]
        # Apply spline weights
        kan_output = torch.sum(basis * self.spline_weights.unsqueeze(0), dim=-1)  # [batch_size, input_size]
        kan_output = kan_output + self.spline_bias
        # Apply dropout
        kan_output = self.dropout(kan_output)
        # Final classification
        logits = self.output_projection(kan_output)
        return logits
class SenToxGLDA(nn.Module):
    """
    Complete SenTox-GLDA model for Chinese online abuse detection
    Combines dual encoders, adaptive fusion, GLDA attention, and KAN classifier
    """
    def __init__(self,
                 sentiment_model_path: str,
                 toxicity_model_path: str,
                 num_classes: int = 2,
                 hidden_size: int = 768,
                 num_heads: int = 12,
                 glda_downsample_ratio: int = 4,
                 dropout: float = 0.1,
                 freeze_encoders: bool = False):
        super().__init__()
        self.num_classes = num_classes
        self.hidden_size = hidden_size
        # Dual encoders
        self.sentiment_encoder = BertModel.from_pretrained(sentiment_model_path)
```

```
        self.toxicity_encoder = RobertaModel.from_pretrained(toxicity_model_path)
        # Freeze encoder parameters if specified
        if freeze_encoders:
            for param in self.sentiment_encoder.parameters():
                param.requires_grad = False
            for param in self.toxicity_encoder.parameters():
                param.requires_grad = False
        # Adaptive fusion module
        self.adaptive_fusion = AdaptiveFusionModule(hidden_size, dropout)
        # GLDA attention mechanism
        self.glda_attention = GLDA(
            hidden_size=hidden_size,
            num_heads=num_heads,
            downsample_ratio=glda_downsample_ratio,
            dropout=dropout,
            fusion_method='gate'
        )
        # KAN classifier
        self.kan_classifier = KANClassifier(
            input_size=hidden_size,
            num_classes=num_classes,
            dropout=dropout
        )
        # Additional layers
        self.dropout = nn.Dropout(dropout)
    def forward(self,
                input_ids: torch.Tensor,
                attention_mask: torch.Tensor,
                return_features: bool = False) -> Dict[str, torch.Tensor]:
        """
        Forward pass of SenTox-GLDA
        Args:
            input_ids: Token IDs [batch_size, seq_len]
            attention_mask: Attention mask [batch_size, seq_len]
            return_features: Whether to return intermediate features
        Returns:
            Dictionary containing logits and optionally features
        """
        # Dual encoder outputs
        sentiment_outputs = self.sentiment_encoder(
            input_ids=input_ids,
            attention_mask=attention_mask,
            return_dict=True
```

```
        )
        toxicity_outputs = self.toxicity_encoder(
            input_ids=input_ids,
            attention_mask=attention_mask,
            return_dict=True
        )
        # Extract hidden states
        sentiment_features = sentiment_outputs.last_hidden_state  # [batch_size, seq_len, hidden_size]
        toxicity_features = toxicity_outputs.last_hidden_state  # [batch_size, seq_len, hidden_size]
        # Adaptive fusion
        fused_features = self.adaptive_fusion(sentiment_features, toxicity_features)
        # GLDA attention
        attended_features = self.glda_attention(fused_features, attention_mask)
        # Apply dropout
        attended_features = self.dropout(attended_features)
        # KAN classification
        logits = self.kan_classifier(attended_features)
        # Prepare output
        output = {'logits': logits}
        if return_features:
            output.update({
                'sentiment_features': sentiment_features,
                'toxicity_features': toxicity_features,
                'fused_features': fused_features,
                'attended_features': attended_features
            })
        return output
    def get_attention_weights(self, input_ids: torch.Tensor, attention_mask: torch.Tensor) -> Dict[str,
torch.Tensor]:
        """Extract attention weights for analysis"""
        with torch.no_grad():
            # Get features up to GLDA
            sentiment_outputs = self.sentiment_encoder(input_ids=input_ids, attention_mask=attention_mask)
            toxicity_outputs = self.toxicity_encoder(input_ids=input_ids, attention_mask=attention_mask)
            sentiment_features = sentiment_outputs.last_hidden_state
            toxicity_features = toxicity_outputs.last_hidden_state
            fused_features = self.adaptive_fusion(sentiment_features, toxicity_features)
            # Extract attention weights from GLDA (would need modification to GLDA class to return weights)
            attended_features = self.glda_attention(fused_features, attention_mask)
        return {
            'fused_features': fused_features,
            'attended_features': attended_features
        }
```

```python
def main():
    """Test SenTox-GLDA model with virtual samples"""
    print("Testing SenTox-GLDA Model")
    print("=" * 50)
    # Model configuration
    config = {
        'sentiment_model_path': 'bert-base-chinese',  # Using base model for testing
        'toxicity_model_path': 'hfl/chinese-roberta-wwm-ext',  # Using base model for testing
        'num_classes': 2,
        'hidden_size': 768,
        'num_heads': 12,
        'glda_downsample_ratio': 4,
        'dropout': 0.1,
        'freeze_encoders': False
    }
    # Create model
    print("Initializing SenTox-GLDA model...")
    model = SenToxGLDA(**config)
    # Model info
    total_params = sum(p.numel() for p in model.parameters())
    trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    print(f"Total parameters: {total_params:,}")
    print(f"Trainable parameters: {trainable_params:,}")
    # Create virtual test data
    batch_size = 2
    seq_len = 128
    # Random token IDs (simulating tokenized Chinese text)
    input_ids = torch.randint(100, 21000, (batch_size, seq_len))  # Chinese vocab range
    attention_mask = torch.ones(batch_size, seq_len)
    # Mask some tokens to simulate real scenarios
    attention_mask[0, 100:] = 0  # First sample has 100 tokens
    attention_mask[1, 110:] = 0  # Second sample has 110 tokens
    print(f"\nInput shapes:")
    print(f"input_ids: {input_ids.shape}")
    print(f"attention_mask: {attention_mask.shape}")
    # Test forward pass
    print(f"\nTesting forward pass...")
    model.eval()
    with torch.no_grad():
        # Basic forward pass
        outputs = model(input_ids, attention_mask)
        logits = outputs['logits']
        print(f"Output logits shape: {logits.shape}")
```

```
    print(f"Logits mean: {logits.mean().item():.6f}")
    print(f"Logits std: {logits.std().item():.6f}")
    # Test with return_features=True
    outputs_with_features = model(input_ids, attention_mask, return_features=True)
    print(f"\nIntermediate features shapes:")
    for key, tensor in outputs_with_features.items():
        if key != 'logits':
            print(f"{key}: {tensor.shape}")
# Test gradient flow
print(f"\nTesting gradient flow...")
model.train()
# Create dummy labels
labels = torch.randint(0, config['num_classes'], (batch_size,))
# Forward pass
outputs = model(input_ids, attention_mask)
logits = outputs['logits']
# Compute loss
loss_fn = nn.CrossEntropyLoss()
loss = loss_fn(logits, labels)
print(f"Loss: {loss.item():.6f}")
# Backward pass
loss.backward()
# Check gradients
grad_norms = []
for name, param in model.named_parameters():
    if param.grad is not None:
        grad_norm = param.grad.norm().item()
        grad_norms.append(grad_norm)
        if 'kan_classifier' in name or 'adaptive_fusion' in name or 'glda_attention' in name:
            print(f"{name}: grad_norm = {grad_norm:.6f}")
print(f"Average gradient norm: {sum(grad_norms) / len(grad_norms):.6f}")
# Test different input lengths
print(f"\nTesting different sequence lengths...")
test_lengths = [64, 256, 512]
for test_len in test_lengths:
    if test_len <= 512:  # Avoid memory issues
        test_input_ids = torch.randint(100, 21000, (1, test_len))
        test_attention_mask = torch.ones(1, test_len)
        try:
            with torch.no_grad():
                test_outputs = model(test_input_ids, test_attention_mask)
                print(f"Sequence length {test_len}: SUCCESS - Output shape {test_outputs['logits'].shape}")
        except Exception as e:
```

```
            print(f"Sequence length {test_len}: FAILED - {e}")

# Test prediction probabilities
print(f"\nTesting prediction probabilities...")
model.eval()
with torch.no_grad():
    outputs = model(input_ids[:1], attention_mask[:1])  # Single sample
    logits = outputs['logits']
    probs = F.softmax(logits, dim=-1)
    print(f"Sample prediction probabilities:")
    for i, prob in enumerate(probs[0]):
        print(f"  Class {i}: {prob.item():.4f}")

# Performance timing
print(f"\nPerformance timing...")
import time
model.eval()
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
if torch.cuda.is_available():
    model = model.to(device)
    input_ids = input_ids.to(device)
    attention_mask = attention_mask.to(device)
    # Warmup
    for _ in range(10):
        with torch.no_grad():
            _ = model(input_ids, attention_mask)
    # Timing
    torch.cuda.synchronize()
    start_time = time.time()
    for _ in range(100):
        with torch.no_grad():
            _ = model(input_ids, attention_mask)
    torch.cuda.synchronize()
    end_time = time.time()
    avg_time = (end_time - start_time) / 100
    print(f"Average inference time: {avg_time * 1000:.2f} ms (GPU)")
    print(f"Peak GPU memory: {torch.cuda.max_memory_allocated() / 1024 / 1024:.2f} MB")
else:
    # CPU timing
    start_time = time.time()
    for _ in range(10):
        with torch.no_grad():
            _ = model(input_ids, attention_mask)
    end_time = time.time()
    avg_time = (end_time - start_time) / 10
```

```
        print(f"Average inference time: {avg_time * 1000:.2f} ms (CPU)")
    print(f"\nSenTox-GLDA model testing completed successfully!")
    print("=" * 50)
if __name__ == "__main__":
    main()
```

train.py

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, get_linear_schedule_with_warmup
from sentox_glda import SenToxGLDA
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score, \
    classification_report
from sklearn.utils.class_weight import compute_class_weight
from tqdm import tqdm
import json
import os
import logging
import argparse
from datetime import datetime
import warnings
import random
from typing import Dict, List, Tuple

warnings.filterwarnings('ignore')
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
class COLDataset(Dataset):
    """Dataset class for COLDataset (Chinese Online abusive Language dataset)"""
    def __init__(self, texts: List[str], labels: List[int], tokenizer, max_length: int = 256):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_length = max_length
    def __len__(self):
        return len(self.texts)
    def __getitem__(self, idx):
        text = str(self.texts[idx])
```

```
        label = self.labels[idx]
        # Tokenize using the same tokenizer for both encoders
        encoding = self.tokenizer(
            text,
            add_special_tokens=True,
            max_length=self.max_length,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }
class SenToxGLDATrainer:
    """Trainer class for SenTox-GLDA model following paper specifications"""
    def __init__(self, config: Dict):
        self.config = config
        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.setup_model()
        self.setup_logging()
    def setup_model(self):
        """Initialize model and tokenizer"""
        # Use BERT tokenizer as the primary tokenizer (compatible with both encoders)
        self.tokenizer = AutoTokenizer.from_pretrained('bert-base-chinese')
        # Initialize SenTox-GLDA model
        self.model = SenToxGLDA(
            sentiment_model_path=self.config['sentiment_model_path'],
            toxicity_model_path=self.config['toxicity_model_path'],
            num_classes=self.config['num_classes'],
            hidden_size=self.config['hidden_size'],
            num_heads=self.config['num_heads'],
            glda_downsample_ratio=self.config['glda_downsample_ratio'],
            dropout=self.config['dropout'],
            freeze_encoders=self.config['freeze_encoders']
        ).to(self.device)
        logger.info(f"Model initialized on {self.device}")
```

```
        # Model statistics
        total_params = sum(p.numel() for p in self.model.parameters())
        trainable_params = sum(p.numel() for p in self.model.parameters() if p.requires_grad)
```

```
        logger.info(f"Total parameters: {total_params:,}")
        logger.info(f"Trainable parameters: {trainable_params:,}")
    def setup_logging(self):
        """Setup logging directory"""
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        self.log_dir = os.path.join(self.config['output_dir'], f'sentox_glda_{timestamp}')
        os.makedirs(self.log_dir, exist_ok=True)
        # Save config
        config_path = os.path.join(self.log_dir, 'config.json')
        with open(config_path, 'w') as f:
            json.dump(self.config, f, indent=2)
    def load_col_dataset(self, data_path: str) -> Tuple[List[str], List[int]]:
        """Load COLDataset following paper specifications"""
        try:
            df = pd.read_csv(data_path)
            # Assuming columns: 'text' and 'label' (0: normal, 1: abusive)
            texts = df['text'].astype(str).tolist()
            labels = df['label'].tolist()
            logger.info(f"Loaded {len(texts)} samples from COLDataset")
            # Log label distribution
            label_dist = pd.Series(labels).value_counts().to_dict()
            logger.info(f"Label distribution: {label_dist}")
            # Handle class imbalance
            unique_labels = np.unique(labels)
            self.class_weights = compute_class_weight(
                'balanced',
                classes=unique_labels,
                y=labels
            )
            self.class_weights = torch.tensor(self.class_weights, dtype=torch.float).to(self.device)
            logger.info(f"Class weights: {self.class_weights}")
            return texts, labels
        except Exception as e:
            logger.error(f"Error loading COLDataset: {e}")
            raise
    def create_data_loaders(self, texts: List[str], labels: List[int]) -> Tuple[DataLoader, DataLoader,
DataLoader]:
        """Create train/val/test data loaders with stratified split"""
        # First split: train+val / test (80% / 20%)
        train_val_texts, test_texts, train_val_labels, test_labels = train_test_split(
            texts, labels, test_size=0.2, random_state=42, stratify=labels
        )
        # Second split: train / val (80% / 20% of train_val)
```

```
        train_texts, val_texts, train_labels, val_labels = train_test_split(
            train_val_texts, train_val_labels, test_size=0.25, random_state=42, stratify=train_val_labels
        )
        # Create datasets
        train_dataset = COLDataset(train_texts, train_labels, self.tokenizer, self.config['max_length'])
        val_dataset = COLDataset(val_texts, val_labels, self.tokenizer, self.config['max_length'])
        test_dataset = COLDataset(test_texts, test_labels, self.tokenizer, self.config['max_length'])
        # Create data loaders
        train_loader = DataLoader(
            train_dataset,
            batch_size=self.config['batch_size'],
            shuffle=True,
            num_workers=self.config.get('num_workers', 4)
        )
        val_loader = DataLoader(
            val_dataset,
            batch_size=self.config['eval_batch_size'],
            shuffle=False,
            num_workers=self.config.get('num_workers', 4)
        )
        test_loader = DataLoader(
            test_dataset,
            batch_size=self.config['eval_batch_size'],
            shuffle=False,
            num_workers=self.config.get('num_workers', 4)
        )
        logger.info(f"Dataset splits - Train: {len(train_dataset)}, Val: {len(val_dataset)}, Test:
{len(test_dataset)}")
        return train_loader, val_loader, test_loader
    def train_epoch(self, train_loader: DataLoader, optimizer, scheduler) -> float:
        """Train for one epoch"""
        self.model.train()
        total_loss = 0
        num_batches = len(train_loader)
        progress_bar = tqdm(train_loader, desc="Training")
        for batch_idx, batch in enumerate(progress_bar):
            # Move to device
            input_ids = batch['input_ids'].to(self.device)
            attention_mask = batch['attention_mask'].to(self.device)
            labels = batch['labels'].to(self.device)
            # Forward pass
            optimizer.zero_grad()
            outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)
```

```
            logits = outputs['logits']
            # Compute weighted loss
            loss_fn = nn.CrossEntropyLoss(weight=self.class_weights)
            loss = loss_fn(logits, labels)
            total_loss += loss.item()
            # Backward pass
            loss.backward()
            # Gradient clipping
            torch.nn.utils.clip_grad_norm_(self.model.parameters(), max_norm=1.0)
            optimizer.step()
            scheduler.step()
            # Update progress bar
            progress_bar.set_postfix({
                'loss': f'{loss.item():.4f}',
                'avg_loss': f'{total_loss / (batch_idx + 1):.4f}'
            })
        return total_loss / num_batches
    def evaluate(self, val_loader: DataLoader) -> Dict[str, float]:
        """Evaluate model on validation/test set"""
        self.model.eval()
        all_predictions = []
        all_labels = []
        all_probabilities = []
        total_loss = 0
        loss_fn = nn.CrossEntropyLoss(weight=self.class_weights)
        with torch.no_grad():
            for batch in tqdm(val_loader, desc="Evaluating"):
                input_ids = batch['input_ids'].to(self.device)
                attention_mask = batch['attention_mask'].to(self.device)
                labels = batch['labels'].to(self.device)
                outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)
                logits = outputs['logits']
                # Calculate loss
                loss = loss_fn(logits, labels)
                total_loss += loss.item()
                # Get predictions
                probabilities = torch.softmax(logits, dim=-1)
                predictions = torch.argmax(logits, dim=-1)
                all_predictions.extend(predictions.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())
                all_probabilities.extend(probabilities[:, 1].cpu().numpy())  # Probability of positive class
        # Calculate comprehensive metrics
        accuracy = accuracy_score(all_labels, all_predictions)
```

```
        macro_f1 = f1_score(all_labels, all_predictions, average='macro')
        weighted_f1 = f1_score(all_labels, all_predictions, average='weighted')
        precision = precision_score(all_labels, all_predictions, average='macro')
        recall = recall_score(all_labels, all_predictions, average='macro')
        # Per-class F1 scores
        f1_per_class = f1_score(all_labels, all_predictions, average=None)
        # AUC score
        try:
            auc = roc_auc_score(all_labels, all_probabilities)
        except ValueError:
            auc = 0.0
        avg_loss = total_loss / len(val_loader)
        metrics = {
            'loss': avg_loss,
            'accuracy': accuracy,
            'macro_f1': macro_f1,
            'weighted_f1': weighted_f1,
            'precision': precision,
            'recall': recall,
            'auc': auc,
            'f1_class_0': f1_per_class[0] if len(f1_per_class) > 0 else 0,
            'f1_class_1': f1_per_class[1] if len(f1_per_class) > 1 else 0,
        }
        return metrics
    def train(self, data_path: str):
        """
        Main training loop following paper specifications:
        - AdamW optimizer (β1=0.9, β2=0.999, ε=1e-8)
        - Learning rate: 2e-5
        - Linear warmup (10% of total steps)
        - Weight decay: 0.01
        - Batch size: 32
        - Sequence length: 256
        - 25 epochs
        """
        # Load data
        texts, labels = self.load_col_dataset(data_path)
        train_loader, val_loader, test_loader = self.create_data_loaders(texts, labels)
        # Setup optimizer following paper specifications
        optimizer = optim.AdamW(
            self.model.parameters(),
            lr=self.config['learning_rate'],
            betas=(0.9, 0.999),
```

```
        eps=1e-8,
        weight_decay=0.01
    )
    # Setup scheduler
    total_steps = len(train_loader) * self.config['epochs']
    warmup_steps = int(0.1 * total_steps)  # 10% warmup
    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=warmup_steps,
        num_training_steps=total_steps
    )
    # Training loop
    best_macro_f1 = 0
    train_history = []
    logger.info("Starting training...")
    logger.info(f"Total epochs: {self.config['epochs']}")
    logger.info(f"Total steps: {total_steps}")
    logger.info(f"Warmup steps: {warmup_steps}")
    for epoch in range(self.config['epochs']):
        logger.info(f"Epoch {epoch + 1}/{self.config['epochs']}")
        # Train
        train_loss = self.train_epoch(train_loader, optimizer, scheduler)
        # Validate
        val_metrics = self.evaluate(val_loader)
        # Log results
        logger.info(f"Train Loss: {train_loss:.4f}")
        logger.info(f"Val Loss: {val_metrics['loss']:.4f}")
        logger.info(f"Val Accuracy: {val_metrics['accuracy']:.4f}")
        logger.info(f"Val Macro-F1: {val_metrics['macro_f1']:.4f}")
        logger.info(f"Val AUC: {val_metrics['auc']:.4f}")
        # Save training history
        epoch_history = {
            'epoch': epoch + 1,
            'train_loss': train_loss,
            **{f'val_{k}': v for k, v in val_metrics.items()}
        }
        train_history.append(epoch_history)
        # Save best model
        if val_metrics['macro_f1'] > best_macro_f1:
            best_macro_f1 = val_metrics['macro_f1']
            self.save_model(os.path.join(self.log_dir, 'best_model'))
            logger.info(f"New best model saved! Macro-F1: {best_macro_f1:.4f}")
        # Save checkpoint every 5 epochs
```

```
            if (epoch + 1) % 5 == 0:
                checkpoint_dir = os.path.join(self.log_dir, f'checkpoint_epoch_{epoch + 1}')
                self.save_model(checkpoint_dir)
        # Final evaluation on test set
        logger.info("Evaluating on test set...")
        test_metrics = self.evaluate(test_loader)
        logger.info("=== Final Test Results ===")
        for metric_name, metric_value in test_metrics.items():
            logger.info(f"Test {metric_name}: {metric_value:.4f}")
        # Save training history and final results
        history_path = os.path.join(self.log_dir, 'training_history.json')
        with open(history_path, 'w') as f:
            json.dump(train_history, f, indent=2)
        final_results = {
            'best_val_macro_f1': best_macro_f1,
            'test_metrics': test_metrics,
            'training_history': train_history
        }
        results_path = os.path.join(self.log_dir, 'final_results.json')
        with open(results_path, 'w') as f:
            json.dump(final_results, f, indent=2)
        logger.info("Training completed!")
        logger.info(f"Best validation Macro-F1: {best_macro_f1:.4f}")
        logger.info(f"Results saved to: {self.log_dir}")
        return final_results
    def save_model(self, save_dir: str):
        """Save model and training state"""
        os.makedirs(save_dir, exist_ok=True)
        # Save model state dict
        torch.save(self.model.state_dict(), os.path.join(save_dir, 'model.pt'))
        # Save tokenizer
        self.tokenizer.save_pretrained(save_dir)
        # Save config
        config_path = os.path.join(save_dir, 'model_config.json')
        with open(config_path, 'w') as f:
            json.dump(self.config, f, indent=2)
        logger.info(f"Model saved to {save_dir}")
    def load_model(self, model_dir: str):
        """Load trained model"""
        model_path = os.path.join(model_dir, 'model.pt')
        self.model.load_state_dict(torch.load(model_path, map_location=self.device))
        logger.info(f"Model loaded from {model_dir}")
def parse_args():
```

```
    """Parse command line arguments"""
    parser = argparse.ArgumentParser(description="Train SenTox-GLDA model")
```

```
    # Data arguments
    parser.add_argument('--data_path', type=str, required=True, help='Path to COLDataset CSV file')
    parser.add_argument('--output_dir', type=str, default='./outputs', help='Output directory for models and
logs')
    # Model arguments
    parser.add_argument('--sentiment_model_path', type=str, default='./models/bert_sentiment_encoder',
                        help='Path to trained BERT sentiment encoder')
    parser.add_argument('--toxicity_model_path', type=str, default='./models/roberta_toxicity_encoder',
                        help='Path to trained RoBERTa toxicity encoder')
    parser.add_argument('--freeze_encoders', action='store_true', help='Freeze pre-trained encoders')
    # Training arguments
    parser.add_argument('--epochs', type=int, default=25, help='Number of training epochs')
    parser.add_argument('--batch_size', type=int, default=32, help='Training batch size')
    parser.add_argument('--eval_batch_size', type=int, default=64, help='Evaluation batch size')
    parser.add_argument('--learning_rate', type=float, default=2e-5, help='Learning rate')
    parser.add_argument('--max_length', type=int, default=256, help='Maximum sequence length')
    # Model architecture arguments
    parser.add_argument('--hidden_size', type=int, default=768, help='Hidden size')
    parser.add_argument('--num_heads', type=int, default=12, help='Number of attention heads')
    parser.add_argument('--glda_downsample_ratio', type=int, default=4, help='GLDA downsampling ratio')
    parser.add_argument('--dropout', type=float, default=0.1, help='Dropout rate')
    # Other arguments
    parser.add_argument('--num_workers', type=int, default=4, help='Number of data loader workers')
    parser.add_argument('--seed', type=int, default=42, help='Random seed')
    return parser.parse_args()
def set_seed(seed: int):
    """Set random seeds for reproducibility"""
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False
def main():
    """Main training function"""
    args = parse_args()
    # Set seed for reproducibility
    set_seed(args.seed)
    # Create config dictionary
    config = {
```

```
        'data_path': args.data_path,
        'output_dir': args.output_dir,
        'sentiment_model_path': args.sentiment_model_path,
        'toxicity_model_path': args.toxicity_model_path,
        'num_classes': 2,
        'epochs': args.epochs,
        'batch_size': args.batch_size,
        'eval_batch_size': args.eval_batch_size,
        'learning_rate': args.learning_rate,
        'max_length': args.max_length,
        'hidden_size': args.hidden_size,
        'num_heads': args.num_heads,
        'glda_downsample_ratio': args.glda_downsample_ratio,
        'dropout': args.dropout,
        'freeze_encoders': args.freeze_encoders,
        'num_workers': args.num_workers,
        'seed': args.seed
    }
    # Log configuration
    logger.info("SenTox-GLDA Training Configuration:")
    for key, value in config.items():
        logger.info(f"  {key}: {value}")
    # Initialize trainer and start training
    trainer = SenToxGLDATrainer(config)
    results = trainer.train(args.data_path)
    logger.info("Training completed successfully!")
if __name__ == "__main__":
    main()
```

5.2 实验数据及相关参考文献

5.2.1 相关实验数据

表 4.1 模型训练对比结果

Model	Precision (%)	Recall (%)	Accuracy (%)	Macro-F1 (%)	AUC (%)
BERT	81.07	78.33	80.37	79.68	88.21
RoBERTa	82.11	79.61	81.90	80.84	89.43
COLDET	83.07	81.33	82.47	81.53	89.07
XLM-R large	78.21	80.00	81.87	79.09	88.43
BAIDUTC	60.66	25.89	63.74	36.29	51.74
omni-moderation-latest	63.71	81.49	75.11	71.51	85.13
MuDA [36]	69.94	72.45	70.28	71.17	80.91
DeepSeek-R1	67.72	82.14	74.41	74.24	86.03
DeepSeek-V3	68.34	**83.33**	76.12	75.09	86.38
Qwen-Plus-Latest	71.29	80.11	76.34	75.44	87.83
Qwen-Turbo-Latest	66.37	79.79	72.88	72.46	85.02
Ours	**83.12**	82.01	**83.24**	**82.56**	**92.14**

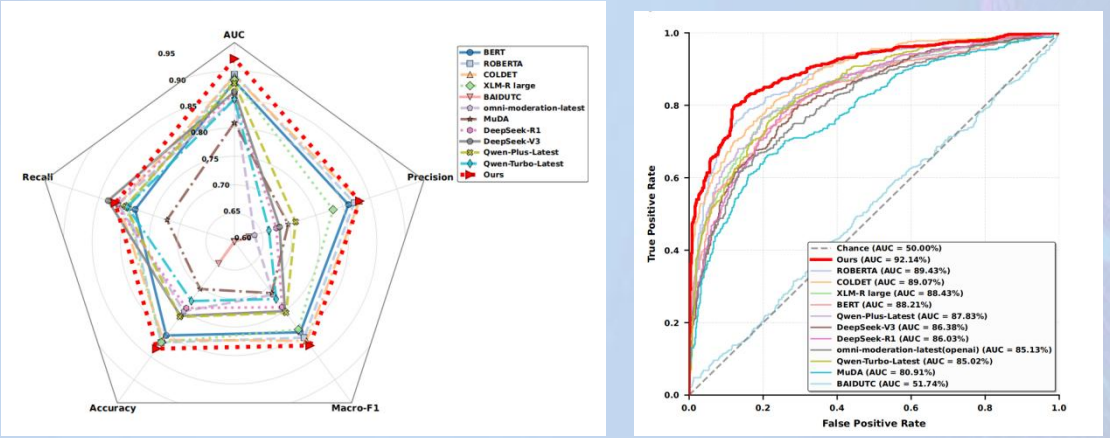

图 4.1 模型训练对比结果

表 4.2 双编码架构效果

BERT Encoder	RoBERTa Encoder	Precision (%)	Recall (%)	Accuracy (%)	Macro-F1 (%)	AUC (%)
✓	✓	**83.12**	**82.01**	**83.24**	**82.56**	**92.04**
×	✓	82.87	81.34	82.75	82.10	90.61
✓	×	70.65	76.81	73.44	73.60	82.45

表 4.3 编码器类型分配影响

Sentiment Encoder	Toxicity Encoder	Precision (%)	Recall (%)	Accuracy (%)	Macro-F1 (%)	AUC (%)
BERT	**RoBERTa**	**83.12**	**82.01**	**83.24**	**82.56**	**92.04**
BERT	BERT	83.00	81.86	83.12	82.43	91.50
RoBERTa	RoBERTa	83.06	81.94	83.17	82.50	91.61
RoBERTa	BERT	83.08	81.99	83.21	82.53	91.72

表 4.4 自适应融合模块有效性

Feature Fusion Type	Precision (%)	Recall (%)	Accuracy (%)	Macro-F1 (%)	AUC (%)
Adaptive Module	**83.12**	**82.01**	**83.24**	**82.56**	**92.04**
Weighted Average	79.66	78.54	80.15	79.09	88.73
Concatenation	79.79	78.51	81.09	79.10	88.33

表 4.5 注意力机制对比

时间表示每个 epoch 的平均训练时间，内存表示训练时的内存使用量

Attention Type	Precision (%)	Recall (%)	Macro-F1 (%)	Time (s/epoch)	Memory (GB)
GLDA (Ours)	**83.12**	82.01	**82.56**	1654	4.612
Multi-Head Attention [37]	78.84	84.32	81.49	1730	4.788
Multi-Head Differential Attention [45]	81.71	81.34	81.52	1717	4.791
Sparse Attention [10]	69.44	**90.27**	78.50	1667	4.035
Tensor Product Attention [48]	80.13	80.12	80.12	1431	**2.985**
Multi-Head Latent Attention [25]	73.35	86.24	79.27	1796	4.978
Concatenation	76.02	74.81	75.14	**1398**	**2.275**

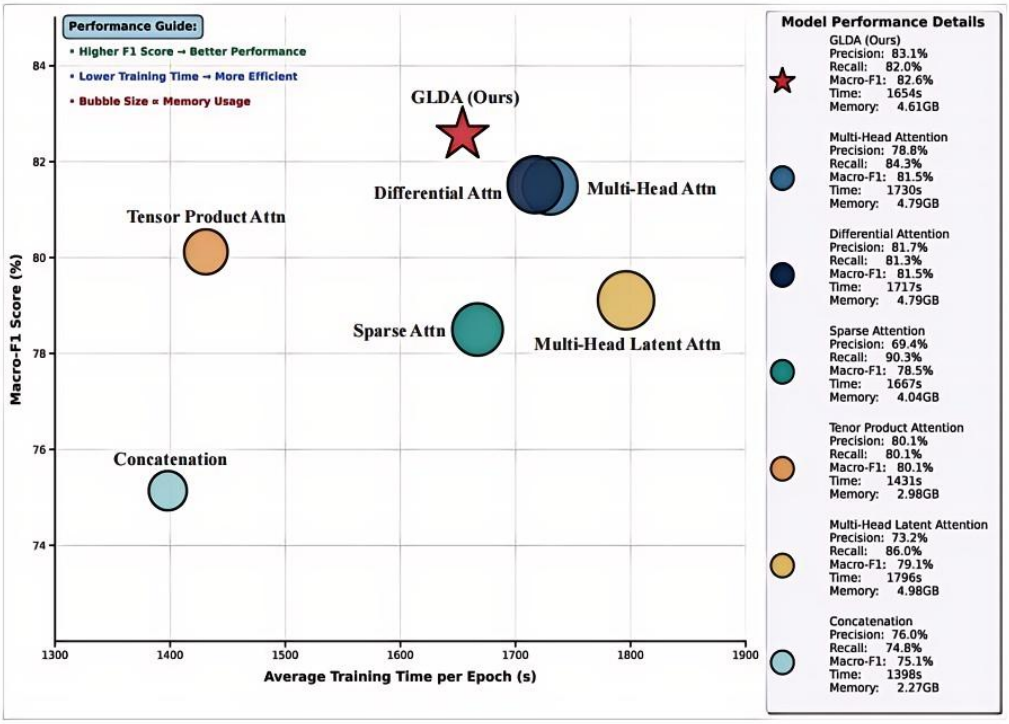

图 4.2 注意力机制的性能对比

表 4.5 GLDA 的双路径比较

Attention Type	Precision (%)	Recall (%)	Macro-F1 (%)	Time (s/epoch)	Memory (GB)
GLDA (Ours)	**83.12**	82.01	**82.56**	1654	4.612
Only LDA	72.53	**87.71**	79.40	**1577**	**2.367**
Only GDA	79.63	81.44	80.52	1627	2.371

表 4.5 GLDA 中多阶差分的影响

Variant	Precision (%)	Recall (%)	Macro-F1 (%)	Time (s/epoch)	Memory (GB)
GLDA (Ours)	**83.12**	82.01	**82.56**	1654	4.612
GLA (No differential)	78.75	79.54	79.14	**1278**	**3.419**
GLFA (No 2nd-order)	80.44	80.17	80.30	1526	4.436
GLSA (No 1st-order)	81.17	80.56	80.86	1598	4.612
GLTA (With 3rd-order)	79.52	**85.41**	82.36	2676	9.747

表 4.5 KAN 变体的比较

Classifier Type	Precision (%)	Recall (%)	Macro-F1 (%)	Memory (GB)
B-Spline Basis KAN	**83.12**	82.01	**82.56**	4.612
Fourier Basis KAN	82.83	82.31	82.55	12.821
Fibonacci Basis KAN	81.71	81.27	81.49	6.368
GRBF Basis KAN	82.42	**82.64**	82.53	9.284
MLP	82.71	80.66	81.67	4.823

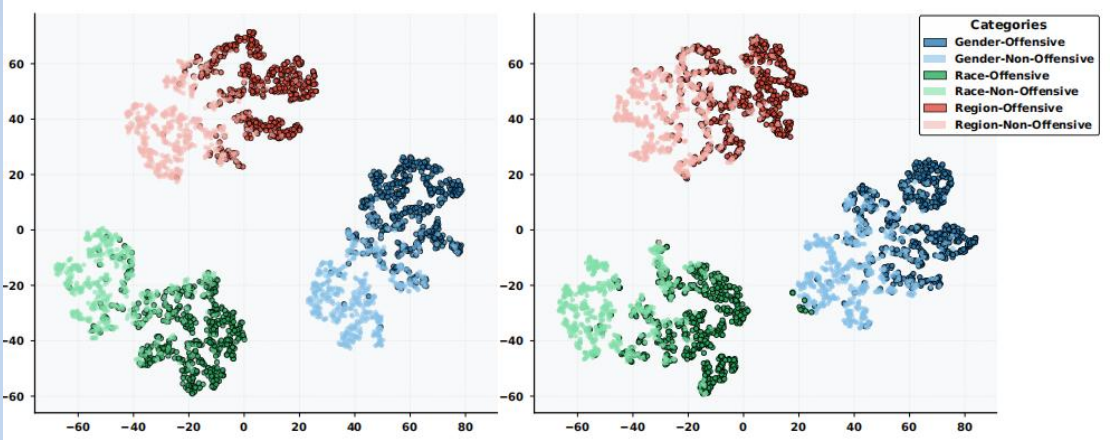

图 4.3

展示了通过 KAN 模型（左）和标准 MLP 分类器（右）对 SenTox-GLDA 数据集倒数第二层进行投影的 3,000 个测试样本可视化结果。与基线模型相比，所提出的 KAN 模型在情感-毒性类别中生成了更紧凑且分离度更高的聚类，显著提升了判别能力，并使决策边界更加清晰。

5.2.2 相关参考文献

[1] M. M. Abdelsamie, S. S. Azab, and H. A. Hefny. The dialects gap: A multi-task learning approach for enhancing hate speech detection in arabic dialects. *Expert Systems with Applications*, 295:128584, 2026.

[2] S. Aggarwal and D. K. Vishwakarma. Exposing the achilles' heel of textual hate speech classifiers using indistinguishable adversarial examples. *Expert Systems with Applications*, 254:124278, 2024.

[3] N. AlDahoul, M. J. T. Tan, H. R. Kasireddy, and Y. Zaki. Advancing content moderation: Evaluating large language models for detecting sensitive content across text, images, and videos. *arXiv preprint arXiv:2411.17123*, 2024.

[4] H. Assoudi. A comparative benchmark of a moroccan darija toxicity detection model (typica. ai) and major llm-based moderation apis
(openai, mistral, anthropic). *arXiv preprint arXiv:2505.04640*, 2025.
[5] Q. Bai, Q. Dan, Z. Mu, and M. Yang. A systematic review of emoji: Current research and future perspectives. *Frontiers in psychology*, 10:476737, 2019.
[6] A. Bleiweiss. Lstm neural networks for transfer learning in online moderation of abuse context. In *ICAART (2)*, pages 112–122, 2019.
[7] S. A. Castaño-Pulgarín, N. Suárez-Betancur, L. M. T. Vega, and H. M. H. López. Internet, social media and online hate speech. systematic review. *Aggression and violent behavior*, 58:101608, 2021.
[8] C. Chen, Z. Xu, Y. Liu, Q. Wu, T. Ji, H. Ji, J. Tang, Z. Sun, L. Fan, J. Liang, et al. Kolmogorov-arnold network for efficient equalization in short-reach im/dd systems. *Optics Express*, 33(16):33139–33152, 2025.
[9] N. Chetty and S. Alathur. Hate speech review in the context of online social networks. *Aggression and violent behavior*, 40:108–118, 2018.
[10] R. Child, S. Gray, A. Radford, and I. Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
[11] L. Cui. Construction and implementation of knowledge enhancement pre-trained language model for text sentiment analysis. *Systems and Soft Computing*, page 200293, 2025.
[12] Y. Cui, W. Che, T. Liu, B. Qin, and Z. Yang. Pre-training with whole word masking for chinese bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3504–3514, 2021.
[13] F. M. P. del Arco, M. D. Molina-González, L. A. Ureña-López, and M. T. Martín-Valdivia. Comparing pre-trained language models for spanish hate speech detection. *Expert Systems with Applications*, 166:114120, 2021.
[14] F. M. P. del Arco, M. D. Molina-González, L. A. Ureña-López, and M.-T. Martín-Valdivia. Integrating implicit and explicit linguistic phenomena via multi-task learning for offensive language detection. *Knowledge-Based Systems*, 258:109965, 2022.
[15] J. Deng, J. Zhou, H. Sun, C. Zheng, F. Mi, H. Meng, and M. Huang. COLD: A benchmark for chinese offensive language detection. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language
Processing*, pages 11580–11599, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics.
[16] M. ElSherief, V. Kulkarni, D. Nguyen, W. Y. Wang, and E. Belding. Hate lingo: A target-based linguistic analysis of hate speech in social media. In Proceedings of the international AAAI conference on web and social media, volume 12, 2018.
[17] M. Fernández-Gavilanes, E. Costa-Montenegro, S. García-Méndez, F. J. González-Castaño, and J. Juncal-Martínez. Evaluation of online emoji description resources for sentiment analysis purposes. Expert Systems with Applications, 184:115279, 2021.
[18] P. Kapil and A. Ekbal. A deep neural network based multi-task learning approach to hate speech detection. Knowledge-Based Systems, 210:106458, 2020.
[19] Z. Li, S. Cao, M. Zhai, N. Ding, Z. Zhang, and B. Hu. Multilevel semantic enhancement based on self-distillation bert for chinese
named entity recognition. Neurocomputing, 586:127637, 2024.
[20] Z. Liu, P. Ma, Y. Wang, W. Matusik, and M. Tegmark. Kan 2.0: Kolmogorov-arnold networks meet science. arXiv preprint arXiv:2408.10205, 2024.

[21] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark. KAN: Kolmogorov–arnold networks. In The Thirteenth International Conference on Learning Representations, 2025.
[22] J. Lu, B. Xu, X. Zhang, C. Min, L. Yang, and H. Lin. Facilitating finegrained detection of Chinese toxic language: Hierarchical taxonomy, resources, and benchmarks. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 16235–16250, Toronto, Canada, July 2023. Association for Computational Linguistics.
[23] Q. Lu, X. Sun, Y. Long, X. Zhao, W. Zou, J. Feng, and X. Wang. Multimodal dual perception fusion framework for multimodal affective analysis. Information Fusion, 115:102747, 2025.
[24] E. Mahajan, H. Mahajan, and S. Kumar. Ensmulhatecyb: Multilingual hate speech and cyberbully detection in online social media. Expert Systems with Applications, 236:121228, 2024.
[25] F. Meng, P. Tang, X. Tang, Z. Yao, X. Sun, and M. Zhang. Transmla: Multi-head latent attention is all you need. arXiv preprint
arXiv:2502.07864, 2025.
[26] M. A. Mersha, M. G. Yigezu, A. L. Tonja, H. Shakil, S. Iskander, O. Kolesnikova, and J. Kalita. Explainable ai: Xai-guided
context-aware data augmentation. Expert Systems with Applications, 289:128364, 2025.
[27] Y. Mu, J. Yang, T. Li, S. Li, and W. Liang. Ha-gcen: Hyperedgeabundant graph convolutional enhanced network for hate speech
detection. Knowledge-Based Systems, 300:112166, 2024.
[28] R. Qian, C. Ross, J. Fernandes, E. Smith, D. Kiela, and A. Williams. Perturbation augmentation for fairer nlp. arXiv preprint arXiv:2205.12586, 2022.
[29] E. B. Ramezani. Sentiment analysis applications using deep learning advancements in social networks: A systematic review. Neurocomputing, page 129862, 2025.
[30] J. Ratkiewicz, M. Conover, M. Meiss, B. Gonçalves, A. Flammini, and F. Menczer. Detecting and tracking political abuse in social media. In Proceedings of the International AAAI Conference on Web and social media, volume 5, pages 297–304, 2011.
[31] D. Sultan, M. Mendes, A. Kassenkhan, and O. Akylbekov. Hybrid cnn-lstm network for cyberbullying detection on social networks
using textual contents. International Journal of Advanced Computer Science and Applications, 14(9), 2023.
[32] H.-T. Ta, D.-Q. Thai, A. B. S. Rahman, G. Sidorov, and A. Gelbukh. Fc-kan: Function combinations in kolmogorov-arnold net
works. arXiv preprint arXiv:2409.01763, 2024.
[33] T. H. Teng, K. D. Varathan, and F. Crestani. A comprehensive review of cyberbullying-related content classification in online social media. Expert Systems with Applications, 244:122644, 2024.
[34] N. Tran, P. Ta, H. Nguyen, H. D. Nguyen, and A.-C. Le. Hybrid contextual and sentiment-based machine learning model for identifying depression risk in social media. Expert Systems with Applications, 291:128505, 2025.
[35] L. van der Maaten and G. Hinton. Visualizing data using t-sne. Journal of Machine Learning Research, 9(86):2579–2605, 2008.
[36] F. Vargas, F. Benevenuto, and T. Pardo. Toward discourse-aware models for multilingual fake news detection. In S. Djabri, D. Gimadi, T. Mihaylova, and I. Nikolova-Koleva, editors, Proceedings of the Student Research Workshop Associated with RANLP 2021, pages 210–218, Online, Sept. 2021. INCOMA Ltd.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need.
Advances in neural information processing systems, 30, 2017.
[38] E. D. Wahyuni, T. L. M. Suryanto, and H. Arviani. Deep learning multimodal sarcasm detection in social media comments: The role of memes and emojis. Journal of Artificial Intelligence and Technology, 5:192–201, 2025.
[39] B. Wang, S. Huang, B. Liang, G. Tu, M. Yang, and R. Xu. What do they “meme”? a metaphor-aware multi-modal multi-task framework for fine-grained meme understanding. Knowledge-Based Systems, 294:111778, 2024.
[40] Z. Wang, D. Huang, J. Cui, X. Zhang, S.-B. Ho, and E. Cambria. A review of chinese sentiment analysis: subjects, methods, and trends. Artificial Intelligence Review, 58(3):75, 2025.
[41] J. Wen, P. Ke, H. Sun, Z. Zhang, C. Li, J. Bai, and M. Huang. Unveiling the implicit toxicity in large language models. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 1322–1338, 2023.
[42] J. Xu, Z. Chen, J. Li, S. Yang, W. Wang, X. Hu, and E. C.-H. Ngai. Fourierkan-gcf: Fourier kolmogorov-arnold network–an effective and efficient feature transformation for graph collaborative filtering. arXiv preprint arXiv:2406.01034, 2024.
[43] L. Xu, R. Mao, C. Zhang, Y. Wang, X. Zheng, X. Xue, and F. Xia. Deep transfer learning model for semantic address matching. Applied Sciences, 12(19):10110, 2022.
[44] Z. Xu. Roberta-wwm-ext fine-tuning for chinese text classification. arXiv preprint arXiv:2103.00492, 2021.
[45] T. Ye, L. Dong, Y. Xia, Y. Sun, Y. Zhu, G. Huang, and F. Wei. Differential transformer. arXiv preprint arXiv:2410.05258, 2024.
[46] R. Yu, W. Yu, and X. Wang. Kan or mlp: A fairer comparison. arXiv preprint arXiv:2407.16674, 2024.
[47] J. Zhang, Y. Fan, K. Cai, and K. Wang. Kolmogorov-arnold fourier networks, 2025.
[48] Y. Zhang, Y. Liu, H. Yuan, Z. Qin, Y. Yuan, Q. Gu, and A. C. Yao. Tensor product attention is all you need. arXiv preprint
arXiv:2501.06425, 2025.
[49] Y. Zhou, P. Xu, X. Wang, X. Lu, G. Gao, and W. Ai. Emojis decoded: Leveraging chatgpt for enhanced understanding in social
media communications. In Proceedings of the International AAAI Conference on Web and Social Media, volume 19, pages 2302–2316,
2025.