成　绩	
评阅人	

复 旦 大 学

课 程 论 文

论文题目：　基于轻量化大模型的饮食分析及建议系统

修读课程：　深度学习及其应用

选课学期：　2025-2026 学年第一学期

选课学生：　黄浩源（23302010090）

完成日期：　2025.12.28

基于轻量化大模型的饮食分析及建议系统

23302010090 软件工程 黄浩源

1 设计背景

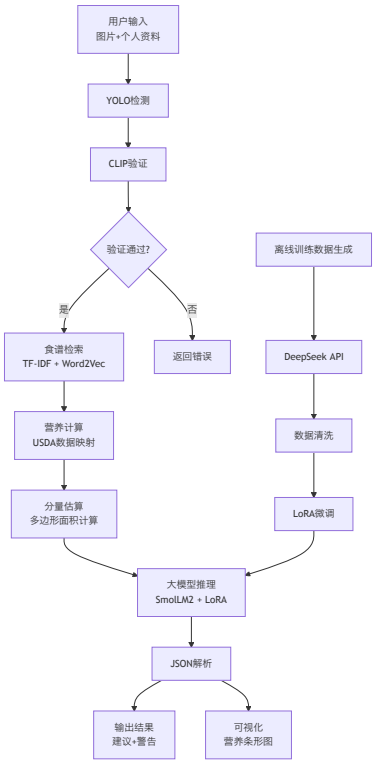
随着社会生活节奏加快，人们对于饮食健康的关注日益增强，但同时也面临着饮食结构不合理、营养摄入不均衡、食物选择困难等问题。传统的饮食管理方式依赖人工记录与营养师咨询，不仅效率低下，也难以实现个性化和实时化。近年来，人工智能与移动计算技术的融合为饮食健康管理提供了新的解决方案。轻量化大模型的出现，使得在资源受限的移动设备上部署智能饮食分析系统成为可能。

本设计旨在结合计算机视觉、自然语言处理与轻量化大模型技术，构建一个端到端的饮食分析及建议系统。系统通过拍摄食物图片，自动识别食物种类、估算分量，并基于用户个人特征与健康目标，提供个性化的营养评估与饮食建议。该系统无需持续联网，保护用户隐私，同时具备高效、准确、易用的特点，适用于日常饮食管理、健康监测、膳食规划等多种场景。

考虑到当今移动设备拥有强大的计算能力与便携性，同时拥有越来越大的本地储存空间，本系统充分利用设备本地资源，实现轻量化部署与实时响应，为用户提供便捷、智能的饮食健康服务。

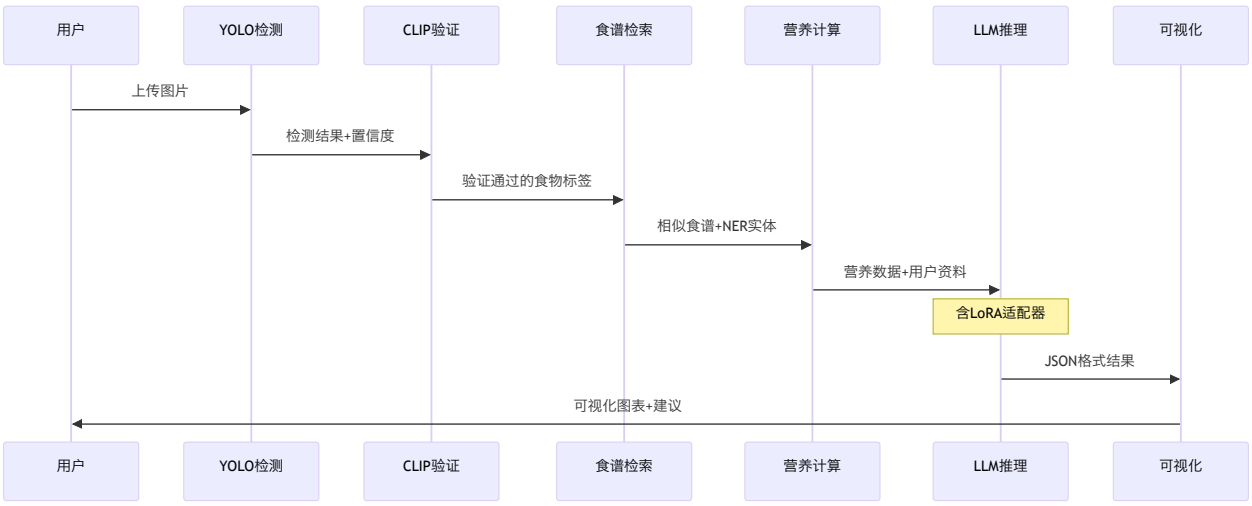
2 系统概览

2.1 架构概览



流程图 2.1: 系统架构概览

2.2 数据流时序概览



时序图 2.2: 数据流时序概览

3 数据准备

3.1 RecipeNLG 数据集

RecipeNLG 数据集（波兹南理工大学）是对 Recipe1M+ 数据集的扩展，提供了更加丰富的食谱资源。与前者不同，该数据集并不着重于将烹饪步骤与对应的图片进行关联，而是强调食谱文本本身的内容、结构与逻辑。经过预处理和去重后，RecipeNLG 收录了超过一百万条全新的食谱，使其成为目前公开可获取的最大规模食谱数据集。数据以 CSV 格式呈现，共有 7 列，分别为

列名	数据类型	备注
id	整数	ID
title	字符串	标题
ingredients	列表或字符串	材料
directions	列表或字符串	步骤
link	字符串	链接
source	ClassLabel	来源
NER	列表或字符串	NER食物实体

表格 3.3: RecipeNLG 数据集格式概览

数据共 2.29 GB，适合作为可移动设备的本地数据库使用。

3.2 USDA Food Central 数据集

FoodData Central 是由美国农业部（USDA） 提供的一个综合性食品成分与营养数据系统，旨在为研究、膳食分析、营养评估和食品开发等领域提供权威、透明且可自由访问的数据资源。它整合了多种类型的食品组成数据，覆盖了从原料、常见食物到品牌食品的详细营养成分信息。在这个任务中我们使用其中的 Foundation Food (JSON, 6.5 MB), SR Legacy (JSON, 205 MB), FNDDS (JSON, 64 MB), Branded (JSON, 3.1 GB) 这四个数据集。

文件	含义
Foundation	基础食品（最权威、结构最规范）
SR Legacy	老版标准参考食品
Survey	膳食调查食品
Branded	品牌食品（包装食品）

表格 3.4: USDA Food Central 数据集分类概览

它们的结构大致如下：

```
Root
 └─ SRLegacyFoods[]                ← 食品数组
     ├─ food metadata              ← 基本信息
     ├─ foodCategory               ← 食品分类
     ├─ foodNutrients[]            ← 营养成分（核心）
     │    ├─ nutrient              ← 营养素定义
     │    ├─ amount                ← 含量
     │    └─ derivation/source     ← 数据来源
     ├─ foodPortions[]             ← 食用分量
     └─ conversionFactors          ← 换算因子
```

我们可以由此导出相关联的营养成分，而且不需要联网。

4 数据处理

4.1 YOLO 输出处理

假设用户用于检测的图片是尽量让盘子占据整个画面的，检测后视觉模型 YOLO 给出的结果如下：

```
{
  "meta": {  // 元数据
    "image_path": "Tempura_01.jpg",       // 图片路径
    "image_size": [1280, 960],            // 图片尺寸（宽×高）
    "model_path": "训练模型路径"            // 使用的YOLO模型
  },
  "objects": [  // 检测到的对象列表（可能有多个）
    {
      "id": 0,                    // 对象ID
      "class_name": "tempura",    // 类别名称
      "class_id": 53,             // 类别ID
      "confidence": 0.864,        // 置信度
      "box": {                    // 边界框坐标
        "x1": 0.99, "y1": 118.19,   // 左上角
        "x2": 942.62, "y2": 1128.87  // 右下角
      },
      "segmentation_poly": [...]  // 分割多边形坐标
    }
  ]
}
```

我们可以从中提取的关键信息包括：

1. **检测到的食物类别**: 稍后会先输入 CLIP, 再用于检索配方；

2. **检测准确度**: 后续会传给大模型
3. **分割多边形坐标**: 用于估计分量
4. **原始图片尺寸**: 作为分量估计的分母

关于分量估计，我们想根据多边形的坐标使用 Shoelace Theorem 计算面积:

$$\text{Area}(\mathbf{x},\mathbf{y}) = \frac{1}{2}\left|\sum_{i=1}^{n}\begin{vmatrix} x_i & y_i \\ x_{i+1} & y_{i+1}\end{vmatrix}\right|$$

其中$\mathbf{x}=(x_1,\cdots,x_n),\mathbf{y}=(y_1,\cdots,y_n)$ $x_{n+1}=x_1,y_{n+1}=y_1$. 而估计的 Portion Ratio 则为

$$\rho = \frac{\text{Area}(\mathbf{x},\mathbf{y})}{w\cdot h} \in (0,1)$$

其中 w 是图片宽度(像素), h 是图片高度(像素)。随后根据用户输入的大致餐盘直径 d，由于我们的假设，我们可以算出 $s=d/w$(与像素的对应比例), 而 Portion Size 可以估计为 $s^2\rho=\frac{d^2\rho}{w^2}$, 单位为 cm^2.

4.2 食谱数据预处理

我们在此将介绍该部分的原理，以及具体的处理方式。

4.2.1 Pickle 化

由于后续我们收集数据进行 LoRA 训练，以及在使用过程中需要重复读取食谱数据，如果重复解析数据 CSV，将不仅会占用 IO，同时也会大大拖慢速度。于是我们选择在第一次读取后储存成 Pickle 二进制文件，第二次读取（重新运行脚本，而非从内存读取）的速度将大大提高。

4.2.2 TF-IDF 向量化

我们将每个食谱的标题、食材以及 NER 实体合并成一个文档，随后按照以下公式计算 TF-IDF (词频逆文档):

$$\text{TF-IDF}(t,d) = \text{TF}(t,d)\cdot\text{IDF}(t)$$

其中 $\text{TF}(t,d)=\dfrac{\text{COUNT}(t,d)}{\sum_{t'\in d}\text{COUNT}(t',d)}$, $\text{IDF}(t)=\log\left(\dfrac{|D|+\delta}{\delta+\sum_{d\in D}\mathbf{1}[t\in d]}\right)+\delta$, D 是文档的集合，即全体食谱;
$\text{COUNT}(t,d)$ 表示词 t 在文档 d 中出现的次数，而平滑化参数 δ 在 scikit-learn 中默认为 1。我们可以由此把文档表示成向量, 记 $w_{t,d}=\text{TF-IDF}(t,d)$, 则文档 $d\in D$ 对应向量表示为

$$\mathbf{v}_d = \begin{pmatrix} w_{t_1,d} \\ w_{t_2,d} \\ w_{t_3,d} \\ \vdots \\ w_{t_n,d}\end{pmatrix}$$

其中 $t_1,t_2,\cdots,t_n\in d$. 归一化(Normalize)后为 $\hat{\mathbf{v}}_d=\dfrac{1}{\|\mathbf{v}_d\|_2^2}\mathbf{v}_d$. 这部分使用 scikit-learn 的 *vectorizer* 简化操作。

4.2.3 用户查询处理

对于某个关键词 q 的查询，我们使用同一个 vectorizer 转换成向量 $\hat{\mathbf{v}}_q$, 然后计算余弦相似度

$$\text{Sim}(p,q)=\frac{\hat{\mathbf{v}}_p^\top\hat{\mathbf{v}}_q}{\|\hat{\mathbf{v}}_p\|_2\|\hat{\mathbf{v}}_q\|_2}=\hat{\mathbf{v}}_p^\top\hat{\mathbf{v}}_q$$

其中 $p\in d, d\in D$. 我们会考虑 Top-k 结果。

4.2.3.1 OOV 处理

4.2.3.1.1 查询失败定义

如果发生 OOV (Out of Vocabulary) 的情况：如果相似度过低，或者查无实体，我们采用 Word2Vec 查询相似词。我们首先定义“查询失败”：

$$\max_{p \in D} \mathrm{Sim}(p, q) < \tau$$

即如果最大相似度小于某个阈值 $\tau > 0$(作品中$\tau = 0.2$)，则认为该查询在文档的离散词空间中缺乏足够的统计证据。

4.2.3.1.2 词向量映射与查询扩展

Word2Vec 将词映射到向量空间中，能找到语义相似的词。例如，对于 OOV 词汇“quinoa”（若不在 TF-IDF 词汇表中），Word2Vec 可以找到类似“rice”或“barley”来扩展查询。在完成分词后, 对于 $Q = \{t_1, \cdots, t_m\}$ 中每个 OOV 的 t_i, 我们提取嵌入向量 $\mathbf{e}_{t_i}$ 然后计算相似度：

$$\mathrm{WordSim}(t_i, s) = \frac{\mathbf{e}_{t_i}^\top \mathbf{e}_s}{\|\mathbf{e}_{t_i}\|_2 \|\mathbf{e}_s\|_2}, \ \forall s \in V_{w2v}$$

其中 V_{w2v} 是 Word2Vec 的词表，我们选取

$$S_{t_i} = \left\{ s | s = \arg\max_s \mathrm{WordSim}(t_i, s), \mathrm{WordSim}(t_i, s) > \sigma, s \in V_{\text{TF-IDF}} \right\}$$

随后扩展 $E = \bigcup S_{t_i}$, 若 $E = \emptyset$, 则回退(fallback)到默认营养值。若非空，则 $q' = \mathrm{join}(Q \cup E)$, 即去重, 随后重试 TF-IDF。

4.2.3.2 计算营养

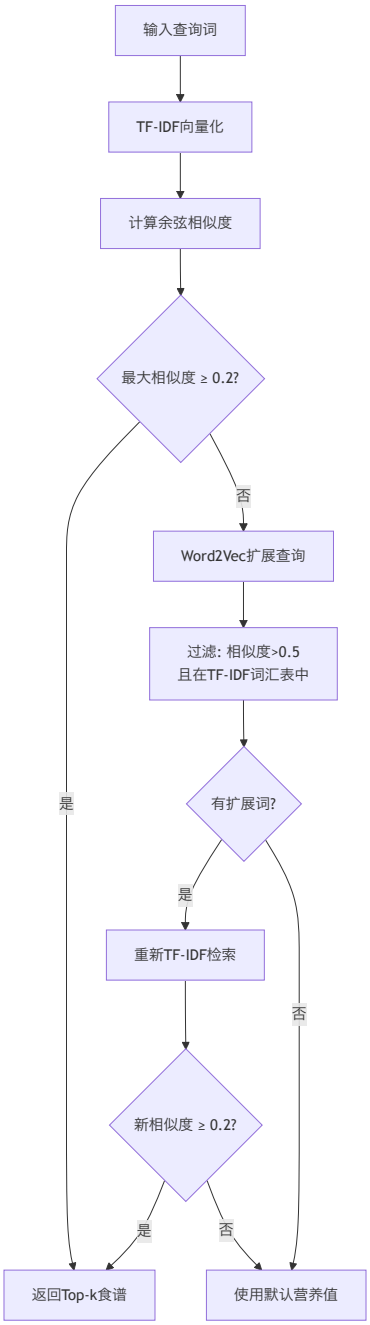
对于匹配文档 $\{d^{(j)}\}_{j=1}^{k}$, NER 实体 E_j, 营养键 k:

$$\mathrm{Avg}_k = \frac{1}{C} \sum_{e \in E_j \cap \mathrm{keys(nutr\_map)}} \mathrm{nutr\_map}(e)[k]$$

其中 nutr_map 代表 *nutrition map*, 而 C 定义如下：

$$C = \sum_{j=1}^{k} |E_j \cap \mathrm{keys(nutr\_map)}|$$

若 $C = 0$，则回退到默认营养值。此部分流程概览如下：



流程图 4.2: 食谱数据以及用户查询处理流程

> 注：我们不可以直接采用 Word2Vec 代替 TF-IDF, 因为若能使用 TF-IDF 进行匹配，那么其精确度必定更高。

4.3 营养数据处理

USDA FoodData Central 数据集分为四个部分，我们需要合并并转换这四个部分的数据，生成一个更易于处理的 JSON 文件。我们从 USDA FoodData Central 四个数据源中，提取每种食物的热量、蛋白质、脂肪、碳水，按食物名称 → 营养信息的形式合并成一个统一的 JSON 文件，最终得到一个如下的映射：

```
{
  "apple, raw": {
    "calories": 52,
    "protein": 0.26,
    "fat": 0.17,
    "carbs": 13.81
  },
  ...
}
```

我们以食物描述为主键，提取营养素。我们遇到描述相同时的合并逻辑为“后面数据覆盖前面数据”，而优先级决定如下：

$$\text{Foundation} > \text{SR Legacy} > \text{Survey} > \text{Branded}$$

5 大模型应用

考虑到移动场景，以及我们需要模型回复的数据量并不大，也并不会过于复杂，所以无需使用付费的大模型 API 来实现这部分的功能。于是我们采用 HuggingFaceTB / SmolLM2-360M-Instruct 模型，是一个针对资源受限环境而设计的紧凑模型，占用硬盘大小 < 1 GB，很适合本地化部署。使用 4 位量化情况下，模型可以更节省空间。

```
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,  # 计算使用半精度
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
)
```

5.1 信息准备

在让大模型给出相应结果之前，我们需要定义用户 Prompt, 即我们需要给模型提供的信息，主要数据字段如下表。

5.1.1 用户个人信息

此部分为用户自行填写。

字段	数据类型	说明
基本信息		
年龄 (age)	integer	整数值
性别 (gender)	string	枚举值："male" 或 "female"
体重 (weight_kg)	integer	单位：千克 (kg)
身高 (height_cm)	integer	单位：厘米 (cm)
活动水平 (activity_level)	string	枚举值： • "sedentary"（久坐） • "lightly_active"（轻度活动） • "moderately_active"（中度活动）
健康目标 (health_goals)	dictionary	包含以下键值： • "goal"：string 类型，枚举值："weight_loss"（减重）、"maintain_weight"（维持体重）、"gain_muscle"（增肌） • "daily_calorie_target"：integer 类型，每日卡路里目标
限制 (constraints)	dictionary	包含以下键值： • "allergies"：string 类型，用户输入的过敏源信息

5.1.2 食物相关信息

食物的营养数据包括卡路里、蛋白质、脂肪、碳水化合物，这些数据通过 TF-IDF 以及 Word2Vec 从食谱数据库 JSON 中检索计算得出。食物的分量比例通过 YOLO 检测到的食物面积（即视觉模型给出的检测结果）计算。

5.1.3 期望的输出

我们期望大模型直接返回一个正确的 JSON，内容如下。

```
{
  "estimated_calories": "估算的总卡路里",
  "condition": "食物对用户目标的适合程度",
  "advice": "具体的营养建议",
  "warnings": ["警告列表"]
}
```

5.2 LoRA 训练

由于基础与训练模型无法给出比较详细的回答，我们决定用 Python 脚本生成数据，然后通过 DeepSeek API 收集回答数据，生成问答对 (q, a)，用于训练 LoRA。我们选择秩为 16 的矩阵，缩放参数 16，目标模块为 `q_proj`，`k_proj`，`v_proj`，`o_proj`. 新的参数公式为

$$W' = W_{\text{原始}} + \Delta W$$

其中 $\Delta W = BA$, $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times k}$, 而 $r \ll \min(d, k)$. 监督式微调所使用的是交叉熵损失，即

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log(P(x_t|x_{<t}; \theta))$$

其中 θ 是模型参数(包含 LoRA 矩阵的参数), x_t 是 t 时刻的 token, $x_{<t}$ 是之前的 token 序列, 而序列长度为 T. 接下来我们通过 `apply_chat_template()` 函数将数据转换为模型期望的对话格式，范例如下：

```
<|im_start|>system
{system_prompt}<|im_end|>
<|im_start|>user
{user_prompt}<|im_end|>
<|im_start|>assistant
{assistant_response}<|im_end|>
```

参数配置如下：

```
    sft_config = SFTConfig(
        output_dir="models/checkpoints",
        num_train_epochs=3,
        per_device_train_batch_size=2,
        gradient_accumulation_steps=8,
        learning_rate=2e-4,
        logging_steps=50,
        save_steps=500,
        max_length=2048,
        optim="adamw_torch",
        report_to="none",
    )
```

输入格式示例如下：

```
{
  "instruction": "Provide personalized nutrition advice...",
  "input": {
    "detected_food": "pizza",
    "portion_ratio": 1.2,
    "recipe_nutrition": {...},
    "user_profile": {...}
  },
  "output": {
    "estimated_calories": 450,
    "condition": "suitable",
    "advice": "...",
    "warnings": [...]
  }
}
```

其中 `input` 对应训练对(q, a)的 q, 而 `output` 则对应训练对(q, a)的 a. 训练的损失曲线如下。该曲线数据读取子SFTTrainer: `logs = trainer.state.log_history`, 可见训练过程相对稳定。

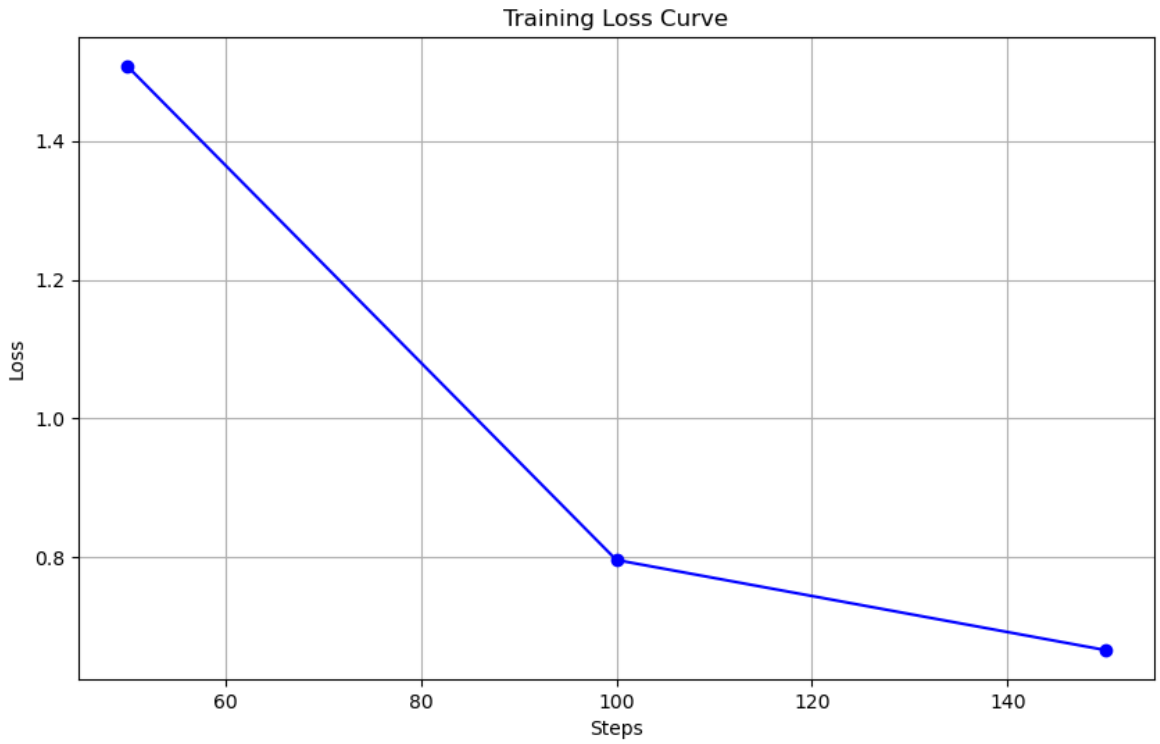


图 5.2: SFTTrainer 给出的训练记录

6 营养参考图生成

营养土 (Nutrition Graph) 用于可视化食物营养成分的分布，通过图形化方式展示主要营养（卡路里、脂肪、碳水化合物、蛋白质等）的含量关系，用于参考。

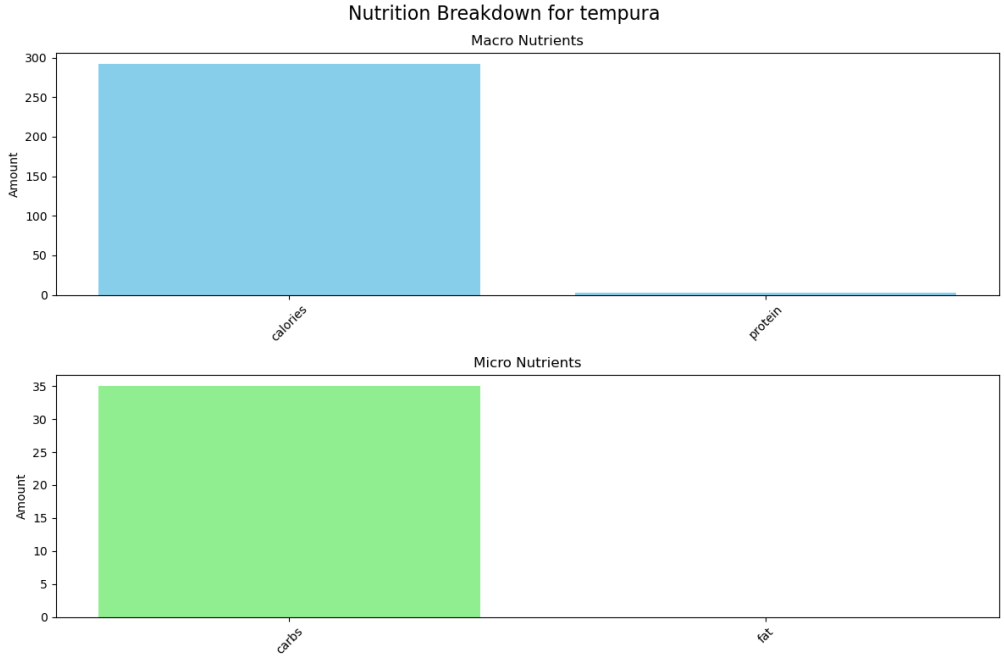


图 6: 营养参考图

7 CLIP 验证

CLIP 验证在本项目中作为 Gatekeeper 角色，主要目标是

1. **验证 YOLO 检测结果**：确保 YOLO 检测到的食物确实存在于图片中；
2. **用户验证描述**：如果用户提供了描述，验证图片是否匹配用户描述；
3. **提高系统鲁棒性**：防止 YOLO 误检导致的错误分析；
4. **语义一致性检查**：确保检测标签与图片内容语义一致

其基本架构为图像编码器和文本编码器的“双塔结构”，共享特征空间。在获得 YOLO 的检测数据后，我们构建待验证文本描述：

```
texts = [f"a photo of {yolo_label}"]  # 例: "a photo of tempura"

if user_desc:
    texts.append(f"a photo of {user_desc}")  # 例: "a photo of fried shrimp"
```

而因为“*a photo of*”是 CLIP 训练时的常见提示格式，我们加入这个描述可以提高模型理解。推理部分代码如下。

```
# 处理输入
inputs = clip_processor(
    text=texts,
    images=image,
    return_tensors="pt",
    padding=True
)

# 移至模型所在设备
device = next(clip_model.parameters()).device
inputs = {k: v.to(device) for k, v in inputs.items()}

# 前向传播
outputs = clip_model(**inputs)
# outputs.logits_per_image 形状: [1, num_texts]
```

在推理结束后，我们计算概率分布：

```
# 计算概率分布
probs = outputs.logits_per_image.softmax(dim=1)[0]
# 例: probs = [0.8, 0.2] 或 [0.3, 0.7]

best_idx = probs.argmax().item()  # 最高概率的索引
best_prob = probs[best_idx].item()  # 最高概率值
```

我们在此定义了一个概率阈值 $\tau = 0.6$, 如果未达到阈值，那么就是验证失败；否则验证成功，返回文本。

8 部署模型

8.1 单样例测试

我们输入一张天妇罗的图片，得到的结果使用 `main.py` 分析：

图 8.1: 天妇罗测试图

8.1.1 大模型回复


```
{
  "estimated_calories": 291.5,
  "condition": "suitable",
  "advice": "This tempura recipe provides approximately 291 calories, which is within your daily calorie target of 2000 for weight maintenance. It offers a moderate protein content (3.3g) and low fat (0.0g) due to the use of vegetable oil. The carbs (35g) are relatively low, but you should consider portion size to ensure it fits within your daily limit. The low sodium content (0.0g) is also beneficial for your health. Given your moderate activity level and allergy to nuts, this tempura is a suitable option for you. However, consider adding a protein source like grilled chicken or tofu to balance the meal. For a vegetarian low-sodium diet, you may want to explore alternative options such as tempura with a lean protein, like tofu or tempeh, and a side of steamed vegetables.",
  "warnings": [
    "Check the recipe for additional ingredients like mayonnaise or breadcrumbs, which could contain nuts.",
    "Monitor your sodium intake, as tempura often uses vegetable oil with high sodium content.",
    "Consider consulting a healthcare professional or registered dietitian for personalized nutrition advice, especially if you have diabetes or hypertension."
  ]
}
```

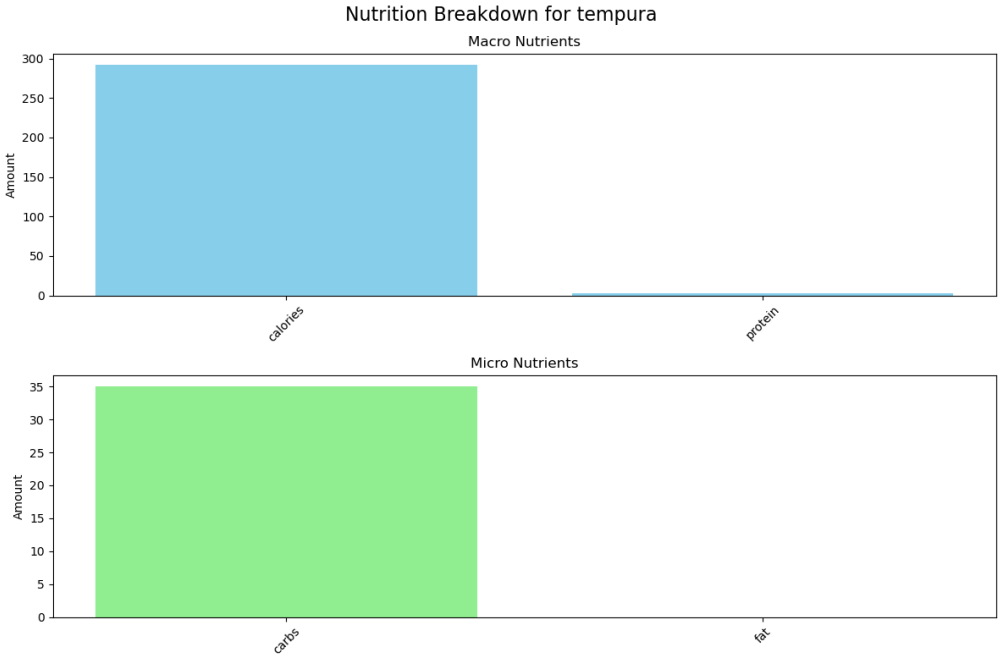



图 8.1.1: 营养参考图

8.2 分析脚本

我们创建了一个分析脚本来带动所有功能。脚本 `nutrition_analyzer.py` 提供了与项目中 `main.py`(测试单个数据用)类似的功能，用于从食物图像中分析营养信息。它使用 YOLO 检测、CLIP 验证，

8.2.1 命令行使用

```
python nutrition_analyzer.py --plate_diameter 20.0 [其他参数]
```

其他参数描述如下(都是可选):

参数	描述
--yolo_json	YOLO 输出 JSON 路径
--user_json	用户配置文件路径
--csv_path	食谱 CSV 路径
--nutr_map_path	营养映射 JSON 路径
--image_path	食物图片路径
--pickle_path	预处理 DataFrame 的 Pickle 路径
--word2vec_path	Word2Vec 模型路径
--llm_name	大模型模型名称
--clip_name	CLIP 模型名称
--lora_path	LoRA 模型路径
--output_graph	营养图输出路径
--hf_token	Hugging Face Token

8.2.2 通过 Import 调用

在其他 Python 脚本中，我们可以通过 `import` 来导入并使用:

```
from nutrition_analyzer import analyze_food

# 调用函数
result = analyze_food(
    image_path="data/MyFood.jpg",
    user_json="data/my_user.json",
    plate_diameter=20.0,
    # 其他参数可选，使用默认值
)

print("Analysis Result:", result)
```

完整函数签名如下，类似于上述命令行调用的参数。

```
def analyze_food(
    yolo_json: str = DEFAULT_YOLO_JSON,
    user_json: str = DEFAULT_USER_JSON_PATH,
    csv_path: str = DEFAULT_CSV_PATH,
    nutr_map_path: str = DEFAULT_NUTR_MAP_PATH,
    image_path: str = DEFAULT_IMAGE_PATH,
```

```
    pickle_path: str = DEFAULT_PICKLE_PATH,
    word2vec_path: str = DEFAULT_WORD2VEC_PATH,
    llm_name: str = DEFAULT_LLM_NAME,
    clip_name: str = DEFAULT_CLIP_NAME,
    lora_path: str = DEFAULT_LORA_PATH,
    plate_diameter: float = None,
    output_graph: str = DEFAULT_OUTPUT_GRAPH,
    hf_token: str = None,
) -> Dict[str, Any]
```

8.3 内存使用分析

根据相关数据，我们可以得到下表的结论：

组件	原始大小	4-bit 量化后	节省比例	备注
SmolLM2-360M	**≈0.7 GB**	**0.27 GB**	**~61%**	360M 参数 FP16 → 4-bit≈260-290 MB
LoRA 适配器	–	8 MB	–	秩=16, α=16
CLIP-ViT-B/32	**~0.34 GB**	**~0.17 GB**	**~50%**	原始≈338 MB
食谱数据库	2.3 GB	2.3 GB	0%	Pickle 格式
营养映射	6.5 MB	6.5 MB	0%	JSON 格式
总计	**≈3.34 GB**	**≈2.75 GB**	**~18%**	适合移动设备

表格 8.2: 内存使用分析

由此，由于量化后只占用约 **2.75 GB** 内存空间，这套系统非常适合部署在移动端。

9 系统限制与未来展望

9.1 当前局限性

1. **营养数据不确定性**
   - 数据集西方数据居多，中餐种类繁多，覆盖率有限
   - 烹饪方式对营养的影响未纳入考量（目前关联用户输入）
   - 食物的季节性和产地差异未纳入考量
2. **用户体验限制**
   - 需要用户输入盘子直径等先验信息
   - 缺少与用户运动数据的联动
3. **模型部署挑战**
   - 轻量化模型牺牲了部分精度
   - 大型数据集占用储存空间

9.2 可扩展功能

1. **多模态输入融合**
   - 扫描条码获取食物商品信息
   - 结合 OCR 识别菜单文字
   - 语音输入补充描述
2. **动态学习与个性化**

- 基于用户反馈调整模型
- 学习用户饮食习惯，提供更精准的建议
- 适配区域性饮食文化

3. **高级营养分析**
   - 血糖生成指数预测
   - 过敏原检测与警告
   - 食物相互作用提示

10 结论

本设计成功构建了一个轻量化、实时、隐私安全（无需联网上传）的饮食分析及建议系统。分析计算机视觉（YOLO）模型的数据，结合多模态理解（CLIP）、信息检索（TF-IDF/Word2Vec）与轻量化大模型（SmolLM2）的有机融合，系统实现了从食物图片到个性化营养建议的完整流程。

- **技术集成**：将 YOLO 检测、CLIP 验证、TF-IDF检索与轻量化 LLM 相结合，构建完整的饮食分析流水线，在移动设备上实现了实时推理；
- **隐私保护**：所有处理均在本地完成，无需上传用户数据；
- **资源优化**：通过 4-bit 量化、LoRA微调、Pickle序列化等技术，将总模型大小控制在适合移动设备使用的尺度上；

尽管系统仍有一些局限性，但随着计算设备的进一步普及、AI模型的持续优化以及营养数据的日益完善，此类智能饮食管理系统或将成为人们日常健康管理的重要组成部分。