ChatGPT

FSGR与PMA融合模型技术设计

1. 融合动机与理论基础

FSGR模型优势与瓶颈： FSGR（Fine-grained Semantic-Guided Region）图像字幕模型针对图像特征的“语义鸿沟”问题提出了有效方案。传统Caption模型常用单模态视觉骨干或CLIP等预训练模型抽取视觉特征，然后仅用自注意力在视觉域融合，高层语义关联难以准确捕获[1]。尤其是CLIP虽然提供了强大的全局跨模态对齐空间，但它只在图像-文本全局层面对齐，导致相同对象的局部视觉特征语义不一致[2][3]。FSGR的贡献在于**细粒度跨模态对齐**：通过**patch级对比学习**聚合语义相似的视觉patch，提高局部特征的语义一致性；进一步通过**语义量化**引入语言信号，将增强后的视觉特征映射到离散的文本语义空间，生成“语言桥接”的视觉特征图；最后经细粒度的跨模态交互融合图像patch和对应的语言语义，显著缩小视觉编码与文本描述之间的语义差距[4]。实验证明FSGR能有效缓解视觉特征的语义不足问题，生成更准确的视觉实体和关系描述[5][6]。然而，FSGR仍有瓶颈：它主要依赖单幅图像本身进行语义对齐和注意力分配，对于超出当前图像范畴的上下文知识（例如场景常识、其他样本中的描述模式）无法利用。此外，虽然FSGR引入了区域级别的显式文本提示，但在更宏观的语义层面（如场景级概念、跨图像的语义模式）尚未充分对齐整合。

PMA的核心思想： PMA（Progressive Multi-Granular Alignment）思想旨在**逐步对齐多粒度的视觉语义**，补足单一粒度对齐的不足。具体而言，PMA-Net原型记忆网络[7]引入了**跨样本的原型记忆机制**，让Caption模型可以“关注”来自其它训练样本的语义信息[8]。通过在训练过程中记录并压缩过往样本的注意力激活，PMA-Net将历史的Key/Value分布建模为一组**原型向量**（prototype），这些原型既具判别性又紧凑[9]。在生成描述时，模型能够对这些原型记忆执行注意力查询，相当于参考“自身过去的经验”来帮助当前图像描述。这实现了一种**多粒度的语义对齐**：除了局部的图像区域-词语对齐之外，模型还在更高层次上对齐了全局语义模式（例如常见场景的描述套路）。研究表明，引入这种原型记忆注意力可以显著提升Transformer字幕模型性能（在COCO数据集CIDEr指标上提升约3.7分）[10]。因此，PMA提供了一个融合**宏观先验**与**微观细节**的途径：宏观上通过跨样本原型获取语义先验知识，微观上通过逐步对齐局部特征实现精细语义关联。

融合动因与预期改进： 将PMA思想融入FSGR，目标是在FSGR细粒度语义对齐的基础上，再引入更高级别的语义先验，从而形成**“渐进式、多粒度”的对齐机制**。FSGR解决了视觉特征与语言语义在局部层面的对齐问题，但仍缺乏利用跨图像全局知识的能力；而PMA提供了跨样本语义模式的记忆，可以补充这一全局视野。二者融合的理论动因在于：**局部精准语义**与**全局语义先验**的结合有望使模型同时“看清这棵树”和“认清整片森林”。具体预期改进包括：提升图像描述的完整性和准确性（既准确识别具体对象和属性，又符合更广泛场景常识）、提高描述的多样性和人类风格（通过记忆参考常见表述）、以及增强开放域泛化能力（CLIP提供开放词汇，记忆模块提供跨域经验）[11][12]。简而言之，融合后的模型将在视觉-语言对齐上**更深更广**：深到细粒度patch与词汇对齐，广至跨样本语义模式对齐，从而生成更语义一致且信息丰富的图像字幕。

2. 网络结构与模块说明

整体架构概览： 融合模型采用Encoder-Decoder框架，新增**PMA-Align模块**贯穿于编码器和解码器阶段，用于分层次对齐视觉与语义特征。整体流程包括：图像经过视觉Encoder提取多尺度特征，PMA-Align模块在编码过程中对视觉patch赋予文本语义标签并在解码过程中引入原型记忆对齐，最后Decoder生成描述文本。下图给出了模型架构各组件及数据流（Encoder和Decoder之间通过多模态特征交互，Decoder同时连接外部原型记忆库）：

[13] [9]

- **Encoder（视觉编码器）：** 使用预训练的CLIP ViT-B/16模型作为图像编码器 [14]。输入图像首先被划分为固定尺寸的patch（例如16×16），经过ViT主干得到每个patch的视觉特征表示，以及一个全局的[CLS]图像特征。CLIP ViT-B/16输出每个patch的特征向量维度为512，patch数量约为196（取决于图像尺寸224×224）外加1个全局特征token [15]。Encoder输出的初始视觉特征张量形状约为 `(N_patches+1) × D`（例如197×512）。在FSGR框架中，我们对CLIP视觉特征施加局部语义优化：增加一个**patch对比学习**分支，使语义相似的patch特征在CLIP隐空间中聚拢。具体实现为在Encoder内部新增损失约束，将同一语义实体的patch拉近、不同语义的patch拉远 [16]（该对比损失计算可通过额外的小型投影头实现，不影响Encoder主干输出维度）。通过这一步，Encoder提取的patch特征已经过**语义一致性增强**，为后续模块打下基础。

- **PMA-Align模块（多粒度对齐模块）：** 该模块由**区域语义对齐**和**原型记忆对齐**两部分组成，对应细粒度和粗粒度两个层次。第一部分是在Encoder阶段插入的**区域级文本提示对齐子模块**：利用CLIP的文本编码器预先构建一个**对象概念词典（Object Semantic Codebook）**，收集一系列潜在对象类别和属性词的文本嵌入向量（例如COCO数据集80类标签或扩充的开放词汇） [17] [12]。对于Encoder输出的每个视觉patch特征，我们计算其与词典中各概念向量的相似度，选择**最近的语义概念**作为该patch的文本标签，将对应的CLIP文本嵌入作为该patch的语义表示 [18]。由此，我们将原本纯视觉表示的patch转换成了一个在CLIP公共嵌入空间中**携带语义标签**的表示（例如把一个patch显式标注为“dog”“grass”等）。所有patch对应的文本嵌入集合构成一幅图像的**语言桥接视觉图谱** [19]，其大小与patch数相同，每个token维度512（与视觉特征等维）。接下来，通过Transformer Encoder中的**跨模态注意力机制**实现视觉特征与文本语义的融合对齐：设视觉patch特征序列$V \in \mathbb{R}^{N\times D}$，语义token序列$T \in \mathbb{R}^{N\times D}$（$N$为patch数，$D=512$），我们在Encoder后部堆叠若干个**Cross-Attention模块**，以视觉特征为查询Q、文本语义为键值K/V，计算注意力加权的语义融合特征 [20]。每个Cross-Attention层使用多头注意力（例如8头，每头$d_k=64$使总$D=512$），输出维度与输入相同(保持512)。经过N_s层堆叠的跨模态交互后，得到增强的视觉特征序列$\tilde{V}$，其中每个patch特征已融合了对应文本提示信息 [6]。这一部分实现了**细粒度层面的语义对齐**：Encoder输出不仅包含纯视觉信息，还带有对齐的显式语言语义。

- **Decoder（文本解码器）：**解码器采用Transformer Decoder结构，负责将编码后的融合特征序列转译成自然语言描述。Decoder的隐藏大小与Encoder一致（例如512维），堆叠L层（如6层Transformer解码层）。标准Decoder每层包含自注意力、跨注意力、前馈网络三个子层。在我们的融合架构中，Decoder每层增加一个**原型记忆注意力子层**，并在跨注意力中处理来自双源的信息：

- **跨注意力 (图像+提示特征):** 在融合模型中，Decoder的跨模态注意力需要同时利用**图像视觉特征**和**区域文本提示**。由于Encoder端我们已将patch特征和文本提示对齐在统一空间，这两者可以直接拼接作为注意力的Key/Value输入。也就是说，每层Decoder的跨注意力以当前层Decoder Query（尺寸$1\times D$，对应当前解码步的词位置隐状态）为Q，以Encoder输出的$\tilde{V}$（尺寸$N\times D$，包含视觉+提示信息）为K,V来计算注意力权重 [21]。经过多头注意力（与Encoder类似配置），Decoder获取当前时刻对图像内容的**上下文向量**。这种设计使Decoder能同时关注图像的局部视觉线索和其语义标签，例如在描述某对象时既看到该对象的视觉特征又“读到”它的类别标签，从而更准确地产生对应词汇。
- **原型记忆注意力:** Decoder每层在跨注意力之后，引入一个并行的**记忆查询注意力**机制 [22]。Decoder维护一个外部**原型记忆库**（Prototype Memory Bank），存储了一组经过学习压缩的跨样本语义原型向量。这些原型是在训练过程中动态更新的：模型在处理每个mini-batch时，将batch中图像编码-解码交互产生的Key/Value（通常是Encoder-Decoder注意力中的K/V激活）缓存到临时内存，并周期性对这些高维特征进行聚类，形成若干**记忆原型**代表 [23] [24]。假设记忆库保留M个原型键值对，每个Key/Value也是$D=512$维（与模型嵌入尺寸一致）。在Decoder的记忆注意力子层，当前解码query同时以这M个原型作为K/V执行多头注意力，计算出一个来自**历史经验**的上下文向量。直观地，它在解码每个词时都询问：“在过去类似场景里，人们通常如何描述？” 例如，当Decoder正在生成关于“滑

雪”的描述词汇时，记忆模块可能激活包含“snow山”、“ski板”等模式的原型，从而注入这些相关语义 [25]。记忆注意力输出的向量与跨注意力输出向量在Decoder中融合（可以采用逐元素加和或拼接后投影等方式，原论文PMA-Net使用逐元素加法融合两路注意力输出 [26]）。融合结果再经过FFN得到Decoder这一层的输出表示。经过L层解码堆叠后，最终输出隐藏状态用于预测下一个词的概率分布。**Decoder各层输入输出维度**均为(batch_size×sequence_length×512)，自注意力和跨模态注意力头数例如8，记忆注意力通常与跨注意力采用相同头数以均衡每头维度。通过这种设计，Decoder能够同时从**当前图像内容**和**训练语料记忆**中汲取信息，生成既贴合图像、又合乎常识和多样化的描述。

各模块作用总结：Encoder负责提取视觉基础特征并利用PMA-Align的区域提示机制嵌入细粒度语义；PMA-Align模块横跨编解码过程，保证视觉patch与文本概念逐一对齐，并提供跨样本的语义原型供生成参考；Decoder依据编码特征生成文本，同时通过记忆模块引入数据集级别的语义先验。这样的结构使模型在每一层级都有相应的对齐措施：**编码阶段对齐像素级对象语义**，**解码阶段对齐跨样本语义模式**，共同提高图像与文本的对齐度和字幕生成质量。

主要层的参数设置： 模型使用与原FSGR类似的Transformer配置。Encoder基于CLIP-ViT，固定Patch大小16，输出维度512；在跨模态融合部分，我们叠加N_s层Cross-Attention（例如2-3层）来充分融合区域视觉和文本提示，注意力头数与维度匹配CLIP（如8头，每头64维）。Decoder维持6层结构，每层除标准自注意力（8头）和FFN（隐层2048维）外，增加一个8头的原型记忆注意力。原型记忆库的大小M可根据数据集规模设置，如COCO上每类别/场景若干个，共计几百个原型向量；实际实现中通过滑动窗口对近期batch聚类得到，以保持记忆的新鲜和多样 [27]。Loss方面除caption损失外，还会有patch对比损失和（可选的）跨模态对齐正则项，这在下一章节详述。

3. 数据流路径

以下以数据流顺序描述从图像输入到字幕输出的过程：

1. **图像输入与CLIP编码：** 输入一张图像，经预处理缩放到224×224并送入CLIP图像编码器（ViT-B/16）。CLIP首先将图像切分为$14\times14=196$个patch，加上1个[CLS]位置，共197个token。经过ViT主干，输出每个token的视觉嵌入特征(维度512)。输出包括一个全局图像特征向量和196个局部patch特征向量 [15]。这些特征初始仅包含视觉模式信息。

2. **区域特征提取与MaskCLIP语义监督：** 为了获取更加有语义意义的区域表示，我们对CLIP提取的patch特征应用MaskCLIP的思想进行语义标注和监督 [28]。具体而言，利用MaskCLIP预先提供的**类别语义嵌入**（例如COCO 80类概念的CLIP文本向量） [29]来指导patch的语义分类。每个patch特征与这些类别嵌入计算相似度，选择最高的类别作为其预测标签，并将对应的文本向量附加到该patch（这相当于进行一次“语义量化”操作) [18]。在训练时，可根据图像的分割/检测标注对patch的语义预测进行监督（MaskCLIP提供了这种密集预测的CLIP适配方案），提高patch语义标签的准确性 [28]。经过这一步，每个patch都分配了一个语义类别，以及一个与该类别对应的文本嵌入表示。输出的数据包括：增强的视觉patch特征序列V（形状$N\times512$）和对应的文本语义序列T（形状同$N\times512$，元素取自CLIP文本空间）。例如，一张图中的patch可能多数被赋予“sky”，“grass”，“dog”等标签，各自用相应的嵌入向量表示 [30]。

3. **跨模态特征对齐融合：** 在得到视觉和文本两路特征后，进入PMA-Align模块的第一阶段——**区域级语义对齐**。这里通过Transformer Encoder中的跨注意力层，将V和T融合：视觉patch特征作为Query，文本语义特征作为Key/Value，计算注意力权重并加权求和 [20]。这一机制使每个视觉patch能够“看向”与其相关的文本概念，加权获取语义信息；同时文本token也会与对应视觉特征交互，强化其表示。经过数层交互，输出**语言增强的视觉特征** $\tilde{V}$：其中每个元素既包含原始视觉内容，又携带明确的语义标签信息 [31]。从数据流上看，原本纯图像特征流在这里与文本特征流合并，形成统一的多模态特征图谱。这一步也可理解为生成了图像的“语义示意图”，每个位置对应图像某区域及其语义含义。

4. **解码器输入准备：** Encoder最终输出的是融合特征$\tilde{V}$（大小$N\times512$），连同CLIP全局图像特征（1×512，可视为特殊的patch）一起，作为Decoder的条件输入。Decoder在生成字幕时将对这些特征执行跨模态注意力。为了便于Decoder利用全局图像信息，我们也可将[CLS]全局特征看作一个特殊的Key加入$\tilde{V}$序列的开头。此时Decoder的Key/Value总长度为$N+1$，维度512。

5. **引入PMA原型记忆模块：** 在进入Decoder生成循环之前，模型还维护/更新**原型记忆库**。如果在训练阶段，当前batch的图像通过Encoder-Decoder交互产生了一系列中间激活（例如Encoder输出经过Decoder跨注意力得到的一些Key/Value），这些激活代表了图像-文本对的特定模式。模型将这些激活累积进一个缓冲区，并定期对其聚类，提取若干**代表性原型向量**更新到记忆库中 [27]。在推理阶段，记忆库保持在训练后学到的原型。这些原型可以视为“训练集中常见描述模式”的压缩表示，例如“某人在滑雪”、“一群人在聚餐”的模式向量。

6. **字幕生成过程（上下文建模）：** Decoder以自回归方式生成描述句子。具体步骤如下：

7. **(a) 自注意力建模已有文本上下文：** Decoder在每一步生成一个词。假设已经生成了部分序列，Decoder首先对已生成的词嵌入序列执行自注意力，将序列中已有的语言上下文融合，输出当前时间步的query向量（尺寸512）代表结合上下文的解码状态。
8. **(b) 跨注意力获取图像细节语义：** 接着，Decoder将该query并行地与Encoder输出的$\tilde{V}$序列进行跨模态注意力 [21]。Key/Value包括图像patch融合特征和全局特征（长度$N+1$），注意力结果是一个结合了当前语境下**图像相关区域**的信息向量。由于$\tilde{V}$中每个特征自带语义标签，注意力机制会自动偏向那些语义与当前生成内容相关的patch。例如，当生成句子的主语时，Decoder可能聚焦于图像中被标记为“person”的patch；生成动作谓语时，可能关注标记为相应动作或物体的patch集合。注意力权重在视觉+文本提示特征上进行归一化，使Decoder能够根据需要从图像或提示中提取信息。这一步确保**图像局部细节**与生成上下文对齐，由注意力权重的高低体现对各区域的关注程度。
9. **(c) 记忆注意力获取全局先验：** 紧随图像跨注意力，Decoder对**原型记忆库**执行注意力查询 [25]。当前query向量与记忆库中的M个原型Key计算注意力权重，从记忆Value中聚合得到一个“记忆语义”向量。这个向量代表了模型参考训练集中**类似上下文**时常用的描述模式，提供了一种全局语义指导。例如，在部分句子“**on a snowboard**”之后，记忆可能提供“snow-covered mountains”这样的短语先验，让模型意识到雪景场合通常会提及山脉背景 [32]。记忆注意力输出的向量在语义空间上也是512维，可与(b)步得到的图像注意力向量相加融合。
10. **(d) 词概率计算：** 最后，融合了图像细节和记忆先验的Decoder向量经过前馈网络和softmax层，产生下一词的概率分布。模型采样或选择概率最高的词作为输出词。然后将该词嵌入加入已生成序列，进入下一个解码时刻循环。如此循环直到输出特殊终止符或达到最长长度。

整个生成过程中，Decoder每一步利用**双通道注意力**进行上下文建模：一方面通过加权的视觉特征获取**图像内容的语境**，确保描述不遗漏图中关键实体和关系；另一方面通过记忆注意力引入**数据集级别的语境**，确保用词和表达符合常识和常见模式。两种注意力机制产生的上下文向量在隐空间中相加，实现信息融合 [26]。例如，对于一张“有人在草地上放风筝”的图，Decoder起始可能从图像特征关注到“person”、“kite”、“grass”等区域提示；同时记忆模块提示可能加入“a person **is flying** a kite on **a sunny day**”之类常见短语结构。最终输出的句子将综合这些信息，既正确涵盖图中要素，又符合常见表达。这样的数据流设计保证了**上下文信息的充分利用**：局部视觉-语义细节通过注意力高亮，跨样本知识通过记忆补充，从而生成的字幕在语义对齐、一致性和丰富度上都得到加强。

4. 实现与训练建议

模型结构复用与模块集成： 鉴于现有FSGR模型已成功训练，我们计划在其代码基础上**增量集成**PMA模块，以最大程度复用已有结构。可以保持原FSGR的`Transformer`类框架，扩充其Encoder和Decoder部分：在Encoder中集成区域文本提示对齐，在Decoder中添加原型记忆注意力机制。具体实现上，建议在`models/fsgr/transformer.py`或`encoders.py`中，于Encoder输出阶段增加Cross-Attention层融合文本嵌入；在`decoders.py`中修改Decoder结构，引入一个新子层用于记忆注意力，并相应调整前向传播顺序（自注意力 ->

跨注意力 -> 记忆注意力 -> FFN）。已有的MaskCLIP投影模块(`projection.py`)仍可用于计算patch语义分配的相似度和提供对比学习损失监督[33]。这种改造保持大部分基础模块不变，仅插入新的对齐机制，对原代码影响有限但功能大幅增强。

PMA模块的位置与集成方式： 根据上述架构设计，PMA思想的融合涉及Encoder和Decoder两个阶段： - Encoder端嵌入： 将**区域语义对齐模块**嵌入Encoder。可在Encoder最后一层或若干层加入Cross-Attention操作：以视觉特征为query，文本语义特征为key/value，实现patch到语义的对齐。代码上，可以扩展Encoder层定义，使其支持一个额外的多头注意力（类似Transformer中的Encoder-Decoder注意力机制，但这里Decoder输入为文本词典embedding）。文本语义特征(来自Codebook)在每个batch可离线计算好（如利用CLIP文本编码80类标签得到`text_embeddings.pth`[29]），然后在Forward中将其提供给Encoder层。若不想改变Encoder层定义，也可在`Transformer.forward`中，在得到Encoder原输出后，再显式调用一个Attention模块完成视觉V与文本T的融合（这样改动更直观）。这一模块的位置可以是在Encoder输出到Decoder输入之间的“后处理”，但为了充分对齐语义，推荐集成在Encoder内部使之端到端训练。

- Decoder端嵌入： 将**原型记忆模块**集成Decoder。需修改Decoder每层结构，增加Memory Attention子层。可以参考标准TransformerDecoder的实现，仿照Encoder-Decoder Attention的模式新增Memory Attention：例如在`DecoderLayer.forward`中，先后调用self_attn、enc_attn，再调用memory_attn，然后再FFN。Memory键值可作为Decoder对象的一个属性，在每次`forward`前由外部更新。具体实现中，我们可以定义一个MemoryBuffer类或使用全局变量，在训练时不断更新原型向量；在模型推理时则加载训练完毕后的记忆原型集合。需要注意Memory Attention的输出与原跨注意力输出维度一致，以便相加融合。因此Memory子层可直接返回加权值向量，再与跨注意力结果相加再进FFN。这样，实现上对`decoders.py`的修改主要是：定义MemoryAttention模块，实例化若干头的MultiheadAttention用于记忆查询；在每层forward中调用并融合。由于PMA-Net源码可参考[34][35]（GitHub: aimagelab/PMA-Net）[36]，我们可以借鉴其中Memory模块的实现逻辑。

参数初始化与预训练： - CLIP相关权重： 利用预训练的CLIP权重初始化图像Encoder和文本Encoder部分（文本编码器用于产生Codebook概念嵌入，可直接使用CLIP提供的，不需要参与训练）。为稳定训练，CLIP的大部分参数应**冻结**或使用极低学习率微调[37]。特别地，文本编码器我们在融合模型中只用作生成词典向量，建议完全冻结以保留其开放词汇嵌入能力[38]。CLIP图像Encoder可冻结主干，大概只在对比学习阶段解冻少量层或者加入Adapter模块进行调整[39]。FSGR论文方法中通过添加少量可学习参数实现对patch特征的细调，这部分权重可以从头初始化，余下CLIP权重加载预训练模型并锁定。 - Transformer部分权重： 可沿用原FSGR Transformer初始化（如Xavier初始化）。由于我们增加了新的Attention层和Memory模块，这些**新增参数**（投影矩阵、原型向量等）需要初始化。注意力的投影矩阵可用标准Xavier方法初始化为均匀随机分布，小的标准差保证初始输出接近零均值。Memory原型向量初始可随机填充或以某种启发式（例如随机抽取几个训练样本特征均值）初始化，但实验表明随机初始化经过训练也能收敛。Memory模块的关键在于训练过程中不断更新聚类，初始值影响不大[11]。总之，新模块权重没有预训练可用，统一采用随机初始化策略即可。

训练策略： 鉴于融合模型引入了额外的训练目标和模块，我们推荐采用**分阶段 + 端到端结合**的训练策略[11]：1. **阶段一，FSGR预训练/初始化：** 先单独训练/调整FSGR部分，使Encoder能够输出语义更一致的patch特征。这一步可以使用**局部对比学习目标**对CLIP视觉Encoder进行微调[40]。具体做法是冻结CLIP大部分参数，只在某几层插入Adapter或解冻最后几层，让模型学习将同类patch聚为一类。训练损失包括：图像区域的对比损失L_{con}（如FSGR采用的patch NCE损失），以及基本的字幕生成损失L_{cap}（交叉熵），二者加权求和。由于此时尚未引入Memory模块，可将Memory模块关闭或Memory权重保持不动。经过若干epoch，使模型基本具备FSGR论文报告的性能和语义对齐效果。 2. **阶段二，融合模型联合训练：** 在初始化的基础上，解冻或保持冻结某些部分，然后启用PMA记忆模块，一起训练完整模型。损失函数采用**多任务联合**形式：以字幕交叉熵损失L_{cap}为主，辅助以FSGR的patch对比损失L_{con}，以及可能加入的CLIP空间对齐正则L_{align}（例如约束图像特征与匹配文本特征的余弦相似度更高）[27]。这样做的目的是确保在训练caption的同时，不遗忘先前学到的细粒度对齐特性，并持续强化视觉-文本对齐。每个mini-batch训练步骤中，我们**动态更新**原型记忆：将本batch产生的新的Key/Value加入缓存，对缓存进行聚类更新原型库[27]。为了控制记忆大小和新旧信息平衡，可采用FIFO队列缓存一定数量的近期样本特征，然后K-Means聚类得到固定数量M的原型。需要注意在联合训练早期，Caption模型可能较弱，因此Memory聚类得到的原型质量有限，可以每隔若

干步再更新一次以避免噪声。此外，可设计记忆更新的**渐进启用**机制：例如训练最初几个epoch关闭记忆模块，仅训练区域提示部分；待caption基线稳定后再逐步增加记忆查询的频率和权重。这类似“curriculum learning”，先学习容易的对齐（单图像细粒度），再学习复杂的对齐（跨图像模式）。 3. **优化与调参：** 训练中采用Adam优化器，对不同模块设置不同学习率：新加入的Memory和Cross-Attention模块可以使用较高学习率(如1e-4)，以加速从随机初始化学习；CLIP主干采用极低学习率或冻结(1e-6或0)，防止其跨模态嵌入被破坏；Transformer其余部分用中等学习率(如5e-5)。同时，可以对Caption损失和对比损失设置权重系数，平衡二者对梯度的贡献。早期可侧重对比损失稳定视觉语义，后期逐渐增加Caption损失权重以优化生成质量。训练后期，如需进一步提升指标，可按常规对Caption模型进行CIDEr优化（Self-Critical强化学习），PMA模块在RL阶段继续提供先验，但注意记忆库更新需冻结（防止RL不稳定更新干扰记忆质量）。

与现有FSGR代码集成要点： - 复用`train_transformer.py`训练脚本的整体流程 [41]，在每个iteration增加记忆更新步骤。确保数据loader输出仍为图像和caption对，只是在模型内部增加处理。 - 调整`optim_entry.py`或相关优化设置，支持多任务loss的优化和不同模块分组参数。新增对齐损失需在loss计算处汇总，建议定义一个总loss = $L_{cap} + \lambda_1 L_{con} + \lambda_2 L_{align}$，梯度回传一次完成。 - 引入大型模型组件时注意GPU显存占用，可能需要减少batch size或启用梯度累计。Memory模块聚类可在CPU上完成（不在计算图中）以减轻显存，并将结果送回模型参数。

总的来说，训练策略应以**稳定性优先**：先保证模型基本收敛，再引入新模块逐步提升。在实践中，采用上述分阶段训练的方法，FSGR和PMA模块能够协同训练：对比损失保证局部对齐稳定，记忆模块提供全局先验，二者共同作用下模型更好地对齐视觉内容与生成文本 [27]。这种训练方案在前人的尝试中已验证有效 [42]，我们预计在当前项目上也可重现类似增益。

5. 对现有代码结构的影响评估

将PMA融入FSGR对代码结构的改动主要集中在模型定义和前向过程，训练脚本则仅需小幅改动：

- **模型定义修改：** 如前所述，需要修改`models/fsgr/transformer.py`或相关文件以加入新模块。具体包括：
- 在Encoder部分（可能在`encoders.py`中定义的`TransformerEncoder`类）增加跨模态注意力层。可能实现方式是修改Encoder的forward，使其接受视觉特征和文本特征双输入；或者更简单地，在`Transformer.forward`里先用原Encoder得到视觉特征，再手工调用一个`MultiHeadAttention`模块完成视觉-文本融合。为了清晰起见，可以新建一个类比如`RegionAlignEncoder(TransformerEncoder)`继承原始Encoder，override其forward包括文本对齐过程。这样对原有代码影响较小，只是在实例化模型时使用新的Encoder类。需注意文本词典embedding需提前加载（例如在模型初始化时从文件加载`text_embeddings.pth`矩阵 [43]），并注册为模型缓冲tensor以供Encoder使用。
- 在Decoder部分（`decoders.py`中的`TransformerDecoder`或`DecoderLayer`定义）增加Memory Attention子层。可以扩充DecoderLayer结构，如增加`self.memory_attn = MultiHeadAttention(...)`和对应的LayerNorm。并在`forward`中调用：`x = self.self_attn(x, x, x) -> x = self.cross_attn(x, enc_output, enc_output) -> x = self.memory_attn(x, memory_proto, memory_proto) -> x = self.ffn(x)`。这里`memory_proto`是从Decoder外部传入的一组原型(key/value)，可以将其附加在Decoder对象上（如Decoder.memory = None在初始化，训练时不断更新）。为了实现memory的更新逻辑，或可在`train_transformer.py`中每个iteration之后调用一个函数更新`model.decoder.memory`。因为Memory不需要梯度，可将其包装为`torch.nn.Parameter`(requires_grad=False)或干脆非参数变量。初次实现可简化为：将Memory看作固定查询库，先随机初始化若干向量，验证代码通路通畅，再实现动态更新。修改Decoder涉及面较大，需要严格确保shape匹配（Memory key/value维度512，对应注意力投影矩阵维度），以及在推理阶段Memory模块也能正确使用（可在`eval`模式下禁用更新，仅读取已有原型）。
- 其他模块：MaskCLIP投影(`projection.py`)仍用于生成视觉特征的语义标签和对比学习，需要确认它能与新的Encoder配合良好。可能需要调整其中text类别数量（之前从4585改为80 [44]）以匹配所用词典大小，或进一步扩展词典以涵盖更多概念。`attention.py`模块（如有自定义Attention实现）可能需

要扩充以支持新的Cross-Attention或Memory-Attention，但通常PyTorch自带MultiheadAttention足以处理，只需正确调用。最后，确保在`models/fsgr/__init__.py`中导出了新增加的模块类，使训练脚本能引用。

- **训练脚本和配置调整：** 在`train_transformer.py`中，主要检查并调整以下几点：

- **数据解包：** 确保数据loader提供的batch内容包括图像tensor和caption。例如之前已修正了batch解包为`images, captions = batch[1], batch[3]` [45]。加入PMA后，这部分不变。
- **模型forward调用：** 当使用新模型时，可能需要提供Memory更新/原型的输入。如果Memory更新在模型内部完成，则训练脚本几乎无需改动，仅需在每个epoch结束后保存模型状态时包括Memory。另外可考虑在每个iteration打印/监控记忆库大小或原型分布以便调试。
- **损失计算：** 如上节所述，新增对比损失和可能的对齐正则。需要在训练脚本中获取这些值并和caption loss加权求和。比如，如果模型`forward`返回多个loss分量，可在脚本里`loss = cap_loss + alpha*con_loss + beta*align_loss`。确保优化器`zero_grad()/step()`涵盖所有参数（包括新增模块参数）。
- **参数管理：** 由于我们对不同模块采用分化学习率，训练脚本中应使用Param Group。例如：`optimizer = Adam([{'params': model.encoder.parameters(), 'lr': 1e-6}, {'params': model.decoder.parameters(), 'lr':5e-5}, {'params': model.decoder.memory_attn.parameters(), 'lr':1e-4}, ...])`。尤其记得把Memory相关参数加入optimizer（Memory原型本身若非Parameter则无须，但Memory Attention的投影矩阵需要）。另外调整learning rate scheduler策略，可能需要更长预热或分段调节以适应多任务训练。
- **日志与评估：** 添加Memory模块后，最好在日志中记录一些额外指标，如对比损失下降情况、Memory中原型数目及覆盖的语义类别等，以监控模块有效性。评估阶段，与原FSGR相同使用COCO Caption指标脚本，Memory模块只读不更新，应该不影响评测流程。

综上，代码改动量集中在模型部分，但因为有现成FSGR和PMA-Net参考，实现具有一定**可控性**。通过合理的模块封装，我们可以尽量避免大幅重构。例如，完全可以在保留原FSGR Encoder/Decoder类的前提下，用继承或wrapper方式实现新功能，从而降低集成风险。预计修改的主要文件有：`encoders.py`（跨模态对齐层），`decoders.py`（记忆注意力层），`transformer.py`（整体衔接逻辑），以及`optim_entry.py`（增加新loss项）。这些改动对其他组件如数据加载、评估等无直接影响。参数方面需要新增如`memory_size`, `prototype_dim`, `contrastive_weight`等配置项，以便灵活调节。适当的默认值和配置注释应在`config`或训练脚本中注明，方便后续维护。

通过上述修改，我们将在现有代码基础上得到一个功能扩展的Caption模型，实现FSGR与PMA的有机融合。在维护好原有代码风格与接口的同时，引入新模块提供性能提升和新功能，代码结构仍清晰可控。

6. 可扩展性建议

设计所融合的**细粒度区域对齐**和**跨样本语义记忆**机制，不仅对图像字幕生成有益，也为相关多模态任务提供了借鉴，具有一定扩展潜力：

- **图文检索（Image-Text Retrieval）：** 本模型在Encoder阶段已经将图像表示映射到CLIP文本概念空间（区域级文本提示），这意味着每张图像被表示为一组**显式语义token**集合。这对图文检索是非常有利的：我们可以将图像的区域文本token与查询文本直接在共享嵌入空间中比对，计算匹配分数。例如，一个查询句子提到某对象，模型能迅速定位图像中对应标记该对象的区域，提高检索准确率 [12]。同时，原型记忆模块提供的全局语义先验也可以用于扩展检索的语义范围——通过记忆中存储的常见描述模式，为图像生成更丰富的语义索引。从实现上看，可以将Encoder+PMA对齐部分当作图像编码器，用于提取语义丰富的图像表示，然后与文本描述计算相似度进行检索。相比传统仅基于全局图像特征的检索，这种细粒度多token表示有望提升对细节的匹配，并具备开放词汇能力（因为CLIP词典带来的泛化）。需要注意检索任务中Memory模块的作用可能略有不同：Memory可以看作一种增强的索引结

构，在大规模检索时可存储跨图像的概念原型，加速近似匹配。但在初步应用时，完全可以忽略Memory，仅使用区域提示增强的图像表示即可，这已经类似于RegionCLIP在检索领域的应用[46]。

- **视觉问答（Image-QA/VQA）：** 在VQA任务中，模型需要结合图像内容和自然语言问题来推理回答。我们设计的融合模型在这方面同样具备优势：首先，Encoder产出的语言桥接视觉特征让模型更容易将问题中的词语与图像区域对应起来（因为图像已经以文本token标注，例如问题问“有几只狗？”，模型能马上识别图像中标记为“dog”的patch）。这种**显式对齐**有助于定位关键信息。其次，Memory原型提供了**跨样本的知识**，可以视作一种知识库，储存了训练集中学到的问答模式或常识。例如，记忆中可能有“天空的颜色通常是蓝色”这样的模式，当问到天空颜色时模型可直接调动该知识作答。将本模型用于VQA的思路是：保持Encoder不变，Decoder改为回答生成器（或分类器），Memory模块仍可提供语义先验。训练时用问答对微调，其中区域提示模块确保图像-问题对齐的精确性，记忆模块提供额外提示（特别是开放性问题或需要常识推理的问题）。由于我们的设计完全基于Transformer多模态注意力，扩展到VQA只需在Decoder部分融合问题文本（作为自回归生成的context或通过对问题做一次cross-attention获取条件向量）。模型开放词汇和跨域知识的特性预计将提升其回答的准确性和丰富度。

- **开放域描述和其他任务：** 由于CLIP提供了开放词汇对齐，本模型本身就具有一定跨域泛化能力，正如FSGR论文在Nocaps数据集上证明的那样[47]。Memory模块进一步注入多样语料经验，使模型在面对训练集中未见过的新场景、新概念时，也能调用“相似场景的描述经验”，因此有潜力应用于**开放域图像描述**、**新闻图像解说**等任务，减少人工标注需求。甚至在图文生成（如故事生成）任务中，记忆库可以扩展存储更长的文本模式，让模型产生连贯段落。

- **模块独立扩展：** 我们的设计采用模块化思想，Encoder的区域对齐和Decoder的记忆查询可以视作独立组件。例如，区域文本提示模块本质上是一个可以插入任何视觉Transformer的**语义对齐插件**；原型记忆模块也可在其他序列生成任务中复用，如机器翻译（记忆过往句对以改进一致性）等。因此，将这两个模块解耦为独立库，对于研究**可解释的注意力对齐**、**模型知识库**都有意义。

小结： 通过融合FSGR与PMA，我们获得的模型在多个方面具备扩展性：细粒度的视觉语义表示和跨样本的知识记忆，使其不仅在图像字幕上达到性能提升，在其它视觉-语言任务上也能发挥作用。凭借Transformer架构的通用性，我们只需对输入输出做相应调整，就能将该模型迁移到图文检索、视觉问答等任务上，预期取得比传统模型更优的表现。这种一次设计、多处适用的特性体现了融合架构的价值，为后续多模态AI系统的开发提供了借鉴方向。

参考文献： 本设计方案参考了FSGR论文[1][4]对于细粒度语义对齐的技术细节，以及PMA-Net相关工作[9][10]关于原型记忆融合Transformer的思想。在实施过程中，以当前已跑通的FSGR代码为基线，结合以上新模块进行开发，可逐步验证并优化模型性能。

---

[1][2][3][4][5][6][13][16][20][30][31][46][47] FSGR论文.pdf
file://file_00000000f034720696202c8c75f327a1

[7][8][9][26][32][34][36] With a Little Help from Your Own Past: Prototypical Memory Networks for Image Captioning
https://openaccess.thecvf.com/content/ICCV2023/papers/Barraco_With_a_Little_Help_from_Your_Own_Past_Prototypical_Memory_ICCV_2023_paper.pdf

[10] With a Little Help from your own Past: Prototypical Memory Networks ...
https://huggingface.co/papers/2308.12383

[11][17][18][19][21][22][23][25][37][38][39][40] 融合原型记忆与区域提示的图像字幕生成网络设计方案.pdf
file://file_0000000003587206bbe5d858814db8b5

12 24 27 42 融合原型记忆与区域提示的图像字幕生成网络设计方案.pdf
file://file_00000000c8ac7208bcd19b84d6448b85

14 15 28 29 33 41 43 44 45 FSGR完整的项目交接文档（执行中的错误以及修改）.md
file://file_0000000008d87206964795b17431d36d

35 Prototypical Memory Networks for Image Captioning | Request PDF
https://www.researchgate.net/publication/377426093_With_a_Little_Help_from_your_own_Past_Prototypical_Memory_Networks_for_Image_Captioning