

# EMGAS:一个基于激活扩散的对话式人工智能情景记忆图谱框架

## 1. 轻量级长效记忆的原则性框架

### 1.1. 挑战:弥合参数化记忆与检索式记忆之间的鸿沟

大型语言模型(LLM)的记忆机制是其认知能力的核心,但当前主流方法存在根本性的二元对立。一方面,模型的参数本身构成了其“参数化记忆”,这种记忆蕴含了海量的世界知识,但本质上是静态的<sup>1</sup>。一旦预训练完成,试图通过微调等方式向其持续灌输新知识,往往会导致“灾难性遗忘”(catastrophic forgetting),即模型在学习新信息的同时,会损害甚至覆盖原有的知识<sup>1</sup>。这种更新能力的缺失,使得纯粹的参数化记忆无法满足动态、持续学习的现实需求。

另一方面,检索增强生成(Retrieval-Augmented Generation, RAG)作为一种外部记忆系统,通过从外部知识库中检索相关信息来增强LLM的上下文,从而为模型提供最新的、领域特定的知识<sup>1</sup>。然而,标准RAG系统也存在显著局限。正如HippoRAG论文中所指出的,传统RAG方法将知识源中的每个段落(passage)孤立地编码和检索,难以实现跨段落的知识整合<sup>1</sup>。当面对需要多步推理或关联不同文档中零散信息才能回答的复杂问题时,这种方法便显得力不从心。LightRAG论文进一步批判了这种“扁平化数据表示”(flat data representations)的局限性,指出其无法捕捉实体之间复杂的相互依赖关系,导致生成的答案支离破碎,缺乏深度和连贯性<sup>1</sup>。

因此,当前领域面临的核心挑战是:如何设计一种既具备参数化记忆的推理整合能力,又拥有RAG系统的动态更新特性,同时还能规避两者缺点的长效记忆系统。理想的系统应是动态的、联想式的,并且在计算上是可行的,它需要避免全模型微调的高昂成本和灾难性遗忘风险,同时也要超越简单文本块检索的浅层关联。

### 1.2. EMGAS愿景:一种神经启发、计算高效的替代方案

为应对上述挑战,本文提出一个全新的长效记忆框架——基于激活扩散的情景记忆图谱(**Episodic Memory Graph with Activation Spreading, EMGAS**)。EMGAS的核心愿景是构建一个系统,它既汲取了HippoRAG的神经生物学灵感(即通过索引结构实现联想记忆)<sup>1</sup>,又采纳了LightRAG对计算效率和增量更新的重视<sup>1</sup>,并最终超越二者,提供一个更加轻量、动态且符合认知科学原理的解决方案。

EMGAS明确地摒弃了HippoRAG中复杂的知识图谱(Knowledge Graph)构建流程和计算成本高昂的个性化PageRank(Personalized PageRank, PPR)算法,也避免了LightRAG中相对简单的双层关键词检索机制。取而代之的是三个更为简洁、动态的核心组件:

1. 简化的记忆图谱(**Memory Graph**):以概念(concepts)和话题(topics)为基本节点,而非形式化的(主语-谓语-宾语)三元组。这种结构更易于从非结构化的对话文本中提取和维护。
2. 增量式构建过程:允许新的记忆(对话片段)实时地、原子化地融入图谱,无需对整个图谱进行重建或重新索引,确保了系统的高时效性。
3. 动态检索机制:采用激活扩散(Spreading Activation)算法进行信息检索。该算法直接模拟了生物神经网络中信号传导的过程,是一种更为直接的神经营过程模拟,能够高效地在图谱中探索关联路径<sup>2</sup>。

为了更清晰地定位EMGAS框架的创新之处,下表将其与HippoRAG和LightRAG进行了多维度对比。

表 1: 不同记忆框架的对比分析

| 特性   | HippoRAG                     | LightRAG                       | EMGAS (本方案)                  |
|------|------------------------------|--------------------------------|------------------------------|
| 图谱结构 | 无模式知识图谱(Schemaless KG),基于三元组 | 实体与关系图谱(Entity/Relation Graph) | 概念与话题图谱(Concept/Topic Graph) |
| 检索算法 | 个性化PageRank(PPR)             | 双层关键词检索                        | 激活扩散(Spreading Activation)   |
| 更新机制 | 离线构建,静态索引                    | 增量式联合(Incremental Union)       | 增量式更新,伴随PMI局部重计算             |
| 遗忘机制 | 无                            | 无                              | 基于指数衰减的激活度衰减与图谱剪枝            |

|               |          |          |          |
|---------------|----------|----------|----------|
| 计算复杂度 (在线/离线) | 离线高, 在线高 | 离线中, 在线低 | 离线低, 在线低 |
|---------------|----------|----------|----------|

该对比清晰地表明, EMGAS旨在解决其前身在设计上固有的核心矛盾。HippoRAG追求的是表征的丰富性, 通过构建详尽的知识图谱和运用强大的图算法(PPR)来实现深度的知识整合<sup>1</sup>。这种方法的优点是能够处理复杂的多跳推理问题, 但代价是高昂的离线构建成本和缓慢的在线检索速度, 这与用户对性能的要求相悖。相对地, LightRAG则优先考虑

操作的效率, 它采用更简单的图谱结构和轻量级的检索机制(关键词匹配), 并支持快速的增量更新<sup>1</sup>。这种设计保证了系统的响应速度和可扩展性, 但其检索能力在处理需要微妙、多步联想的查询时可能有所不足。

用户的需求恰恰揭示了对一种“第三条道路”的渴望:既要具备足够的表征丰富性以支持联想推理, 又不能牺牲计算效率。EMGAS的设计理念正是为了开辟这条道路。其核心突破在于认识到, 检索算法本身的动态性可以弥补图谱结构的简化。激活扩散算法是一种动态过程, 它能在-一个相对简单的图谱上通过模拟能量传导来发现多跳关联路径<sup>2</sup>。这使得EMGAS能够以接近LightRAG的结构简单性, 实现HippoRAG所追求的推理能力, 从而完美地解决了表征丰富性与操作效率之间的核心矛盾。

## 2. 情景记忆图谱(EMG):一种动态数据结构

情景记忆图谱(Episodic Memory Graph, EMG)是EMGAS框架的基石。它被设计成一个动态、加权的图结构, 用于编码和存储从对话历史中提取的记忆片段。其设计理念源于对人类情景记忆的模拟, 即记忆并非由孤立的事实构成, 而是由相互关联的概念组成的网络, 这些关联的强度会随着经验的累积而动态变化。

### 2.1. 节点架构: 记忆的基石

EMG的节点架构摒弃了HippoRAG中形式化的(主语, 谓语, 宾语)三元组结构<sup>1</sup>, 采用了更符合对话流特征的、更具认知灵活性的两类节点。

- **概念节点 (Concept Nodes):**这些节点是记忆网络中最基本的原子单位, 代表了信息的核心内容。它们通常是对话中出现的关键词、命名实体或关键短语(例如, “LangChain框架”, “API密钥”, “数据持久化”)。概念节点是连接到原始文本片段的直接桥梁。每个概念节点都存储着一组丰富的元数据, 以便于后续的检索和记忆动态管理:
  - id: 节点的唯一标识符, 通常是概念经过标准化处理(如词形还原)后的文本。

- text: 概念的原始文本。
- source\_passage\_ids: 一个集合, 包含了所有包含该概念的原始对话片段的ID。这使得在检索到节点后, 能够迅速定位到原始上下文。
- timestamp\_created: 节点首次被创建时的时间戳。
- timestamp\_last\_accessed: 节点最近一次在对话中被提及或在检索中被激活的时间戳。
- base\_activation\_score: 节点的基础激活值, 这是一个核心参数, 用于后续的时间衰减和记忆强化计算。
- 话题节点 (**Topic Nodes**): 这些节点代表了更高层次的抽象, 用于组织和聚合语义上相关的概念节点。例如, 概念节点“React”, “Vue”和“Svelte”可能都连接到一个名为“前端框架”的话题节点。话题节点不直接链接到具体的文本片段, 而是作为图谱中的组织中枢。它们的创建是动态的, 基于对共现概念的聚类分析。这种设计引入了一种轻量级的层次结构, 既满足了LightRAG中对低层细节和高层主题信息的检索需求<sup>1</sup>, 又通过一个统一的图谱结构实现了更为整合的表达。

## 2.2. 边架构与联想强度

EMG中的边代表了节点之间的关联, 其设计强调统计上的联想强度, 而非显式的语义关系标签(如“is-a”或“part-of”)。

- 联想边 (**Associative Edges**): 连接两个概念节点的加权、无向边。这些边的权重是EMG的核心, 它量化了两个概念之间的“联想强度”(associative strength)。一个权重较高的边意味着这两个概念在对话历史中存在强烈的、非偶然的关联。
- 层次边 (**Hierarchical Edges**): 从概念节点指向其所属话题节点的有向、非加权边。这些边构建了图谱的层次结构, 使得检索时可以从具体概念泛化到更广泛的话题。

## 2.3. 基于增量式点互信息(PMI)的边加权

如何客观且高效地计算联想强度是图谱构建的关键。EMGAS提出了一种基于点互信息(Pointwise Mutual Information, PMI)的、可增量更新的边加权方案, 这是对HippoRAG中基于嵌入相似度创建同义关系边的直接改进<sup>1</sup>。

PMI是一种成熟的信息论度量, 用于衡量两个事件同时发生的概率与它们各自独立发生概率的乘积之间的差异<sup>4</sup>。在自然语言处理中, 它常被用来度量词语之间的关联度。其数学定义为:

$$\text{PMI}(x,y) = \log(p(x)p(y)p(x,y))$$

其中,  $p(x)$  和  $p(y)$  分别是概念  $x$  和  $y$  出现的概率,  $p(x,y)$  是它们共同出现的概率。一个高的PMI值

表明概念  $x$  和  $y$  的共现并非偶然，而是存在强关联。

在EMGAS中，这些概率将通过在对话历史的滑动窗口（例如，一个完整的对话回合）内统计概念的出现次数和共现次数来估计。为了处理对话数据稀疏性的问题——即很多概念对可能从未共现过，导致  $p(x,y)=0$  从而使PMI值为负无穷——本方案将采用正点互信息（Positive PMI, PPMI）。

PPMI将所有负的PMI值置为0，这使得模型只关注正向关联，对于稀疏数据更为鲁棒<sup>4</sup>。

$$\text{PPMI}(x,y) = \max(0, \text{PMI}(x,y))$$

传统的PMI计算需要遍历整个语料库来获得准确的词频和共现频率，这与用户对增量更新的硬性要求相冲突。为了解决这个问题，EMGAS借鉴了流式计算（streaming algorithms）中对PMI的近似处理方法<sup>7</sup>。系统将在内存中维护一个全局的计数器，用于存储每个概念的出现总次数和每对概念的共现总次数。当一个新的对话片段被处理时，系统仅需更新其中出现的概念及其共现对的计数。随后，仅需重新计算与这些被更新的概念相连的边的PPMI权重。这种局部更新的策略避免了对整个图谱或语料库的全局扫描，极大地提高了更新效率。

这种基于PMI的加权方法，不仅仅是一个技术选择，它从根本上定义了图谱的性质。HippoRAG和LightRAG都试图通过LLM提取显式的语义关系，如（Thomas, researches, Alzheimer's），这是一种对客观事实的建模，旨在构建一个“知识”图谱<sup>1</sup>。然而，这种方法复杂、昂贵且易出错。相比之下，PMI完全基于统计共现。如果“Thomas”和“Alzheimer's”在对话中频繁一起出现，无论它们之间的具体关系是什么，PMI都会赋予连接它们的边一个很高的权重<sup>4</sup>。这恰恰模拟了人类情景记忆的运作方式：我们常常记得两个概念是相关的，因为我们在经验中频繁地将它们联系在一起，即使我们已经忘记了最初将它们联系起来的具体句子。因此，使用PMI构建的图谱不再是一个关于客观事实的“知识图谱”，而是一个反映对话历史中联想模式的“记忆图谱”。这不仅更真实地模拟了记忆，也极大地简化了图谱的构建和维护过程，完全符合用户的设计要求。

## 2.4. 时间维度：新近度与激活

为了实现用户提出的时间加权和遗忘机制，EMGAS为每个节点引入了时间维度。每个节点都拥有一个base\_activation\_score（基础激活值）。这个值在节点创建时被初始化，其初始大小可以由概念提取时的重要性分数（例如，由YAKE算法提供）和创建时间戳共同决定。

这个基础激活值是记忆动态变化的基础。一方面，它会随着时间的推移而自然衰减，模拟记忆的“遗忘”过程。另一方面，当一个节点所代表的概念在新的对话中被再次提及，或在检索过程中被激活时，它的timestamp\_last\_accessed会被更新，同时其base\_activation\_score会得到一次“增强”（boost）。这个过程模拟了通过复习和使用来强化记忆的认知现象。这种设计确保了最近被提及或频繁被提及的记忆比陈旧、不常用的记忆具有更高的基础活性。

# 3. 记忆印痕的构建与增量更新

为了让EMG能够实时地反映对话的进展，必须有一套高效的流程来处理新的对话内容并将其无缝地整合到图谱中。这个过程被称为“记忆印痕”(Memory Engram)的构建，它模拟了生物学中新经验转化为持久记忆痕迹的过程。

### 3.1. 记忆摄入管道

当一个新的对话回合(一个“情景”)发生时，系统会启动一个标准化的记忆摄入管道，将非结构化的文本转化为结构化的图谱更新。该管道包含以下步骤：

1. 输入：一个新的文本块，通常由用户的提问和AI的回答共同组成。
2. 概念提取：应用一个轻量级的、无监督的关键词提取算法，从文本块中识别出候选的概念节点。
3. 概念标准化：对提取出的概念进行标准化处理，如词形还原(lemmatization)或词干提取(stemming)，以确保同一概念的不同变体(例如，“run”、“running”、“ran”)能够映射到同一个图谱节点上。这一步对于避免节点冗余至关重要，并且可以利用现有的TypeScript/JavaScript库轻松实现<sup>10</sup>。
4. 共现统计：在标准化的概念集合中，识别在预定义窗口内(例如，在同一个对话回合内)共同出现的概念对。
5. 图谱更新：执行增量更新算法(详见3.3节)，将新的节点、边的权重调整以及激活度变化应用到EMG中。

### 3.2. 基于YAKE的无监督概念提取

概念提取是记忆摄入管道的第一步，也是至关重要的一步。选择何种提取算法直接影响到整个系统的性能和复杂度。与HippoRAG依赖大型、昂贵的LLM进行开放信息抽取(OpenIE)不同<sup>1</sup>，EMGAS为了满足用户对轻量、高效和低依赖性的要求，选择了\*\*YAKE(Yet Another Keyword Extractor)\*\*算法<sup>14</sup>。

选择YAKE的理由与用户需求高度契合：

- 无监督性：YAKE完全基于文本的统计特征(如词的大小写、位置、频率、与上下文的关联度等)来计算关键词分数，无需任何预先标注的训练数据<sup>14</sup>。
- 语料库独立性：该算法专为处理单个文档而设计，这意味着它可以完美地应用于独立的对话片段，而无需依赖整个对话历史或外部语料库<sup>14</sup>。
- 轻量与高效：YAKE的计算过程不涉及复杂的模型推理，速度非常快，足以满足实时对话处理的性能要求。

- 语言无关性：由于其基于统计特征而非语言学模型，YAKE在理论上可以应用于多种语言，具有很好的通用性<sup>14</sup>。

相比之下，诸如TextRank这类同样是无监督的图排序算法，虽然强大，但需要在每个对话片段内部构建一个临时的词图并进行迭代计算，对于实时性要求极高的场景来说可能过重<sup>18</sup>。而YAKE提供了一种“一次性”计算分数的方法，更适合作为记忆摄入管道的前端。

### 3.3. 增量更新算法

增量更新是EMGAS的核心特性，确保了记忆图谱的动态演化。该算法的设计灵感来源于LightRAG的增量联合思想<sup>1</sup>，但针对EMGAS的PMI加权和时间敏感特性进行了深度定制。以下是该算法的伪代码表示：

TypeScript

```
function incrementalUpdate(graph: MemoryGraph, newTextChunk: string): void {
    // 步骤 1 & 2: 提取并标准化概念
    const concepts = extractAndNormalizeConcepts(newTextChunk); // 使用YAKE和词形还原

    // 步骤 3: 识别共现对
    const cooccurrences = findCooccurrences(concepts, windowSize);

    // 步骤 4.1: 更新节点并强化记忆
    for (const concept of concepts) {
        if (!graph.hasNode(concept)) {
            const initialScore = calculateInitialActivation(concept);
            graph.addNode(concept, {
                timestamp: now(),
                baseActivation: initialScore
            });
        } else {
            // 强化现有记忆: 更新时间戳并提升基础激活值
            graph.boostNodeActivation(concept, { timestamp: now() });
        }
    }
}
```

```

// 步骤 4.2: 更新全局共现计数
for (const pair of cooccurrences) {
    graph.incrementCooccurrenceCount(pair);
}
graph.incrementTotalObservations();

// 步骤 4.3: 局部重计算受影响边的PPMI权重
const affectedNodes = new Set(concepts);
for (const node of affectedNodes) {
    graph.incrementNodeCount(node); // 更新单个概念的计数
    for (const neighbor of graph.getNeighbors(node)) {
        const newPPMI = calculatePPMI(node, neighbor, graph.getGlobalCounts());
        graph.updateEdgeWeight(node, neighbor, newPPMI);
    }
}
}

```

这个算法的精妙之处在于它将全局影响(PMI权重的变化)通过局部计算来实现。每次更新只涉及新文本中出现的概念及其直接邻居，避免了对整个图谱的遍历，从而保证了操作的高效性。

这种增量更新机制创造了一个对话与记忆结构拓扑之间的动态反馈循环。在一个静态索引的系统中，关系一旦建立就基本固定。新信息只是作为新的节点或边被添加，而旧的结构保持不变。然而，在EMGAS中，边的权重(即联想强度)会随着新的对话内容而动态变化。例如，一对最初很少共同出现的概念可能只有一条微弱的连接。但如果后续的对话突然开始频繁地将它们联系在一起，这对概念的共现计数值会迅速增加，导致它们之间的PPMI权重飙升。这相当于记忆图谱根据新的经验“重新布线”，以反映这种新建立的理解。因此，EMGAS的记忆不仅是在增长，更是在不断地自我重组。这种拓扑结构的流动性，比简单地添加孤立事实更能模拟生物学上通过重复和新情境来强化记忆联想的强大机制。

## 4. 通过激活扩散进行联想检索

当需要为新的用户查询生成上下文时，EMGAS采用了一种名为“激活扩散”(Spreading Activation)的检索算法。这个算法被明确设计为HippoRAG中个性化PageRank(PPR)的一种轻量级、神经科学启发的替代方案，它避免了PPR的巨大计算开销，同时又能有效地在记忆图谱中进行多步联想推理。

### 4.1. 从查询到初始激活

检索过程始于对用户当前查询的分析。

1. **查询解析**: 与记忆摄入管道类似, 用户的查询文本首先通过YAKE算法进行处理, 提取出其中的核心概念<sup>14</sup>。
2. **种子节点定位**: 这些从查询中提取出的概念被映射到EMG中对应的概念节点。这些节点将成为激活扩散过程的“种子节点”(seed nodes)。
3. **能量注入**: 系统向这些种子节点注入一股初始的“激活能量”。在算法实现上, 这意味着将它们的瞬时激活值( $A[i]$ )设置为一个较高的初始值(例如, 1.0), 而图谱中所有其他节点的瞬时激活值则保持为0。这个步骤模拟了当一个想法或问题进入大脑时, 与之直接相关的神经元被首先激活的现象, 这与标准激活扩散算法的起始阶段完全一致<sup>2</sup>。

## 4.2. 激活扩散算法: PPR的替代方案

一旦种子节点被激活, 激活能量就开始通过图谱的边网络向外扩散。这个过程是迭代进行的, 旨在发现与初始查询概念高度关联的其他记忆节点, 即使它们之间没有直接连接。该算法的核心流程基于经典的认知模型<sup>2</sup>:

1. **初始化**: 如上所述, 为种子节点设置初始激活值。
2. **迭代**: 在每个迭代步骤中, 所有瞬时激活值 $A[i]$ 超过预设“点火阈值”(firing threshold,  $F$ )的节点都会“点火”。
3. **传播**: 一个点火的节点*i*会将其一部分激活能量传递给它的所有邻居节点*j*。能量的传递量由以下核心公式决定:

$$A_{new}[j] = A_{old}[j] + (A[i] \times W[i,j] \times D)$$

其中:

- $A[i]$  是点火节点*i*当前的激活值。
- $W[i,j]$  是连接节点*i*和*j*的边的权重, 即它们之间的PPMI值, 代表了联想强度。
- $D$  是一个“传播衰减因子”(propagation decay factor), 是一个介于0和1之间的值, 用于模拟信号在传递过程中的能量损失。
- $A_{new}[j]$  和  $A_{old}[j]$  分别是邻居节点*j*更新后和更新前的激活值。

4. **终止**: 扩散过程在满足以下任一条件时终止:
  - 达到预设的最大迭代次数。这是一个重要的工程控制, 用以保证检索的响应时间。
  - 整个图谱中总的激活值变化量低于某个微小的delta阈值, 表明系统已达到一个相对稳定的激活状态。

HippoRAG论文中强调了一个传统RAG系统难以解决的关键挑战, 即“路径发现”(path-finding)问题<sup>1</sup>。例如, 当查询“哪位斯坦福教授研究阿尔茨海默症?”时, 如果没有任何一个文档同时提及“斯坦福”、“阿尔茨海默症”和“Thomas Südhof”教授, 传统检索就会失败。PPR通过模拟从“斯坦

福”和“阿尔茨海默症”这两个种子节点开始的随机游走来解决这个问题；游走路径最可能交汇的节点（即“Thomas Südhof”）会获得最高的排名。

激活扩散算法以一种更直接、更符合神经直觉的方式实现了同样的目标。当激活能量同时从“斯坦福”和“阿尔茨海默症”两个种子节点向外扩散时，一个与这两个节点都有强连接（即高PPMI权重边）的节点，如“Thomas Südhof”，将会从两个不同的路径接收到激活能量。根据传播公式中的累加效应( $A_{new}[j] = A_{old}[j] + \dots$ )，“Thomas Südhof”节点累积的激活值会自然地高于那些只与其中一个种子节点相连的节点。因此，激活扩散不仅是PPR的一个计算上更简单的替代品，它本身就是一种优雅的机制，内在地解决了驱动HippoRAG采用复杂图算法的“路径发现”问题。

表 2：激活扩散超参数指南

为了消除用户对“过多超参数”的顾虑，下表对激活扩散算法中的关键参数进行了详细说明，将其从神秘的“旋钮”转变为透明的、可控制的“杠杆”，用于平衡检索过程的广度（探索）和深度（利用）。

| 参数           | 描述               | 典型范围      | 调优影响  |
|--------------|------------------|-----------|---|
| 初始激活值        | 注入到种子节点的初始能量。    | 1.0       | 通常固定，作为能量传播的起点。                                   |
| 点火阈值 ( $F$ ) | 节点传播其能量所需的最小激活值。 | 0.0 - 1.0 | 较高的值导致更集中的、探索性较弱的搜索；较低的值则允许更广泛的、更远的联想。            |
| 传播衰减 ( $D$ ) | 能量在每跳传播中损失的比例。   | 0.0 - 1.0 | 较高的衰减值（接近0）将搜索限制在种子节点的近邻；较低的衰减值（接近1）允许激活传播到更远的节点。 |
| 最大迭代次数       | 强制终止扩散过程的硬性限制。   | 3 - 10    | 直接控制检索延迟。较少的迭代次数返回更直接的关联，更多的迭代次数则能发现更深层次的间接关联。    |

|                             |                      |       |                                |
|-----------------------------|----------------------|-------|--------------------------------|
| 时间衰减Lambda<br>( $\lambda$ ) | (见第5节) 控制记忆背景性遗忘的速率。 | $> 0$ | 影响节点的基础激活值, 从而间接影响其在检索中被激活的潜力。 |
|-----------------------------|----------------------|-------|--------------------------------|

### 4.3. 从激活的记忆痕迹中组装上下文

当激活扩散过程终止后, EMG呈现出一幅激活景观图, 其中不同节点根据其与查询的关联度拥有了不同的激活值。

1. 节点排序与选择: 系统识别出最终激活值最高的N个节点。这些节点代表了与查询最相关的概念, 其中既包括查询中直接提及的概念, 也包括通过多跳联想发现的间接相关概念。
2. 源文本定位: 系统收集这些顶级节点的元数据中存储的source\_passage\_ids。
3. 上下文构建: 使用这些ID从一个简单的文档存储(如键值对存储或文档数据库)中检索出原始的对话文本片段。这些片段随后被拼接在一起, 形成最终的、丰富的上下文, 并被传递给LLM用于生成最终的答案。这个最终步骤与标准RAG流程的生成阶段相符<sup>1</sup>。

## 5. 记忆动态: 遗忘与巩固

一个真正模拟人类记忆的系统不仅需要学习和联想, 还需要能够忘记。遗忘并非缺陷, 而是一种关键的优化机制, 它能防止信息过载, 并使系统能够优先处理重要和相关的记忆。EMGAS通过两种互补的机制——被动的激活衰减和主动的图谱剪枝——来实现这一动态过程。

### 5.1. 基于指数衰减的遗忘数学模型

为了满足用户对记忆衰减功能的要求, EMGAS引入了一个全局的、周期性运行的遗忘函数。该函数模拟了记忆随着时间的推移而自然淡化的被动过程。其核心数学模型借鉴了认知科学中的记忆衰减曲线, 特别是指数衰减模型<sup>21</sup>。

该函数会作用于图谱中每个节点的base\_activation\_score(基础激活值)。其更新公式为:

$$A_{\text{new}} = A_{\text{old}} \times e^{-\lambda \Delta t}$$

其中：

- $A_{old}$  是节点当前的基础激活值。
- $A_{new}$  是衰减后新的基础激活值。
- $\lambda$  ( $\text{lambda}$ ) 是一个正的衰减常数，这是一个可调的超参数，用于控制遗忘的速度。较大的  $\$\\lambda$  值意味着更快的遗忘。
- $\Delta t$  是自上次衰减计算以来经过的时间。

这个公式简洁而强大，它确保了所有记忆痕迹都会持续不断地衰减其“能量”，除非得到外部的强化。该公式易于在TypeScript中实现，并且只引入了一个关键的超参数 $\$\\lambda$ ，符合用户对算法简单性的要求。

## 5.2. 图谱剪枝与巩固

随着时间的推移，对话中可能充满了大量短暂的、无关紧要的概念。如果这些概念永久地留在图谱中，即使它们的激活值很低，也会导致图谱规模不必要地膨胀，从而影响整体性能。为此，EMGAS引入了一种主动的图谱维护机制——剪枝。

系统会周期性地（例如，在系统低负载时段）运行一个剪枝算法。该算法会遍历图谱中的所有节点和边，并删除那些基础激活值已经衰减到预定义的一个极低阈值以下的节点。当一个节点被删除时，所有与之相连的边也会被一并移除。

这个过程模拟了对琐碎、不重要信息的永久性遗忘，起到了两个关键作用：

- 性能优化：保持图谱的规模在一个可管理的范围内，确保检索和更新操作的效率。
- 记忆巩固：通过移除噪音，使得图中保留下来的都是经过时间和使用考验的、具有较高激活值的“强”记忆，从而使图谱的结构更加清晰地反映出核心的、长期的知识关联。

这两种机制——在访问时增强激活（见2.4节）和全局性的激活衰减——的相互作用，使得系统能够自然地涌现出短期记忆和长期记忆的区别。当一个新概念首次进入图谱时，它获得一个初始激活值。如果这个概念之后再也未被提及或检索，指数衰减函数将持续降低其激活值，直至其最终被剪枝算法清除。这完美地模拟了短期记忆因未能得到巩固而消失的过程。相反，如果一个概念被频繁地在对话中提及或在检索中被激活，它会反复地获得激活值的“增强”。这种增强会持续地对抗衰减效应。久而久之，那些被频繁强化的概念将能维持一个较高的基础激活值，使其能够抵抗遗忘，并长久地保留在图谱中。这模拟了记忆通过重复和应用从短期存储转化为长期存储的巩固过程。这个动态平衡使得EMGAS能够自动地、无监督地学习哪些信息是重要的（长期记忆），完全基于其在对话中的使用模式，这是一种极其强大且高效的记忆管理机制。

## 6. TypeScript实现蓝图

本节提供了一个具体的、面向实践的实现蓝图，旨在将前述的理论框架转化为可直接在Node.js环境中用TypeScript编写的代码。蓝图包括核心数据结构的定义、关键算法的伪代码实现，以及对持久化策略的讨论。

### 6.1. 核心数据结构

以下是用于构建EMG的核心TypeScript接口和类结构的建议。这些定义清晰、类型安全，并直接反映了模型的设计。

TypeScript

```
// 定义激活扩散算法的选项
interface SpreadingActivationOptions {
    firingThreshold: number;
    propagationDecay: number;
    maxIterations: number;
}

// 定义记忆图谱中节点的数据结构
interface MemoryNode {
    id: string; // 节点的唯一标识符，例如，词形还原后的概念
    type: 'concept' | 'topic'; // 节点类型
    baseActivation: number; // 基础激活值，受时间和访问影响
    createdAt: Date; // 创建时间戳
    lastAccessed: Date; // 最后访问时间戳

    // 仅概念节点拥有
    sourcePassageIds?: Set<string>; // 关联的原始文本片段ID
}

// 定义记忆图谱中边的数据结构
interface MemoryEdge {
    sourceId: string;
```

```
targetId: string;
weight: number; // 边的权重, 即PPMI分数
}

// 核心的MemoryGraph类, 封装了所有操作
class MemoryGraph {
    // 使用Map来高效地存储和查找节点与边
    private nodes: Map<string, MemoryNode>;
    private edges: Map<string, Map<string, MemoryEdge>>; // 邻接表示法

    // 用于增量计算PMI的全局计数器
    private conceptCounts: Map<string, number>;
    private pairCounts: Map<string, number>; // key为 "concept1|concept2"
    private totalObservations: number;

    constructor() {
        this.nodes = new Map();
        this.edges = new Map();
        this.conceptCounts = new Map();
        this.pairCounts = new Map();
        this.totalObservations = 0;
    }

    // 公共API方法
    public incrementalUpdate(textChunk: string): void {
        // 实现第3.3节中的增量更新逻辑
    }

    public retrieveContext(query: string, options: SpreadingActivationOptions): string {
        // 实现第4章中的激活扩散检索逻辑
        // 返回的是需要被提取的原始文本片段的ID数组
    }

    public applyDecay(lambda: number): void {
        // 实现第5.1节中的指数衰减逻辑
    }

    public pruneGraph(threshold: number): void {
        // 实现第5.2节中的图谱剪枝逻辑
    }

    //... 其他私有辅助方法, 例如:
    // private extractAndNormalizeConcepts(text: string): string {...}
}
```

```
// private calculatePPMI(concept1: string, concept2: string): number { ... }  
// private boostNodeActivation(nodeId: string): void { ... }  
}
```

## 6.2. 算法实现伪代码

以下是MemoryGraph类中两个核心方法的详细伪代码，其逻辑清晰，可直接翻译为TypeScript代码。

### retrieveContext (包含激活扩散)

代码段

```
function retrieveContext(query, options):  
    // 步骤1: 初始化  
    seedNodes = extractAndNormalizeConcepts(query)  
    activations = new Map()  
    for node in graph.nodes:  
        activations.set(node.id, 0.0)  
  
    for seed in seedNodes:  
        if graph.hasNode(seed):  
            activations.set(seed, 1.0)  
  
    // 步骤2: 迭代扩散  
    for i from 1 to options.maxIterations:  
        firedNodes =  
        for node in graph.nodes:  
            if activations.get(node.id) > options.firingThreshold:  
                firedNodes.push(node)  
  
        if firedNodes is empty:  
            break // 提前终止  
  
        newActivations = new Map(activations)  
        for firingNode in firedNodes:  
            currentActivation = activations.get(firingNode.id)
```

```

neighbors = graph.getNeighbors(firingNode.id)
for neighbor in neighbors:
    edgeWeight = graph.getEdgeWeight(firingNode.id, neighbor.id)
    energy = currentActivation * edgeWeight * options.propagationDecay

    // 累加能量
    newActivations.set(neighbor.id, newActivations.get(neighbor.id) + energy)

    // 点火后，自身能量衰减(可选策略，可防止能量无限循环 )
    newActivations.set(firingNode.id, currentActivation * (1 - options.propagationDecay))

activations = newActivations

// 步骤3: 收集结果
sortedNodes = sort nodes by activations in descending order
topNodes = take top N from sortedNodes

passagelids = new Set()
for node in topNodes:
    if node.type is 'concept':
        for id in node.sourcePassagelids:
            passagelids.add(id)

return Array.from(passagelids)

```

### 6.3. Node.js环境下的持久化与可扩展性

对于一个真实世界的应用，记忆图谱必须被持久化，以防止在服务重启时丢失所有记忆。根据应用规模和性能要求的不同，可以采用不同的持久化策略。

- 内存数据库与持久化（适用于中小型应用）
  - 推荐方案：[Redis](#)<sup>25</sup>。
  - 原因：Redis是一个极其快速的内存数据存储，其丰富的数据结构与图谱模型能很好地映射。例如，可以使用Redis Hashes来存储节点的属性（baseActivation, lastAccessed等），使用Redis Sets来存储每个节点的邻居列表（即邻接表）。这种方式可以实现极低的读写延迟。
  - 持久化策略：Redis提供了两种主要的持久化机制。RDB（快照）可以周期性地将整个内存中的数据集保存到磁盘。AOF（Append-Only File）则会记录每一个写操作命令。两者结合使用，可以在保证高性能的同时，确保数据在服务重启后能够恢复。对于EMGAS来说，这意味着整个MemoryGraph对象的状态可以被安全地保存和加载。

- 文档数据库(适用于大规模部署)
  - 推荐方案: **MongoDB**<sup>25</sup>。
  - 原因: 当图谱的规模大到无法完全容纳于内存时, 基于磁盘的数据库是必需的。相比于关系型数据库, MongoDB的灵活文档模型更适合存储半结构化的图数据。
  - 持久化策略: 可以采用邻接表示例模式(**Adjacency List Pattern**)。在这种模式下, MongoDB中的一个集合(collection)用于存储所有节点, 每个文档代表一个节点。文档中除了包含节点的属性外, 还包含一个数组字段(如neighbors), 该数组存储了所有与该节点直接相连的邻居节点的ID以及对应边的权重。例如:

```
JSON
{
  "_id": "lemmatized_concept_A",
  "type": "concept",
  "baseActivation": 0.85,
  //... 其他属性
  "neighbors":
}
```

- 这种模式虽然在进行深度图遍历时可能需要多次查询, 但对于激活扩散这种通常限制在少数几步迭代内的算法来说, 性能是完全可以接受的。

选择何种持久化策略会直接影响增量更新算法的具体实现。对于Redis这样的内存数据库, 更新计数器和边权重是原子性的、毫秒级的操作(例如, 使用HINCRBY和HSET命令)<sup>25</sup>。整个更新过程可以同步完成。而对于MongoDB, 由于涉及磁盘I/O, 频繁的单文档写入可能会成为性能瓶颈。因此, 在MongoDB的实现中, 可能需要采用批处理或异步更新的策略。例如, 可以先将计数更新批量写入, 然后触发一个后台任务来异步地重新计算并更新受影响边的PPMI权重。这个架构上的权衡将直接影响到系统的实时性能和实现复杂度。

## 7. 结论与未来工作

### 7.1. EMGAS框架总结

本文详细阐述了一个为对话式人工智能设计的长效记忆框架——EMGAS。该框架旨在解决现有RAG系统在知识整合、计算效率和记忆动态性方面的不足, 提供了一个轻量、高效且符合认知科学原理的解决方案。通过综合分析, EMGAS框架的核心优势可总结如下:

- 轻量级: 采用简化的概念/话题图谱结构, 避免了构建和维护复杂知识图谱的高昂成本, 使其更易于从非结构化对话中实时构建。

- **高效性**:核心的增量更新算法确保了新记忆能够以局部计算的方式无缝融入图谱。同时，采用计算成本远低于PPR的激活扩散算法进行检索，保证了在线查询的低延迟。
- **动态性**:引入了基于指数衰减的遗忘机制和基于访问的记忆强化机制。这使得图谱中的联想强度能够随对话历史动态演化，并能自动区分和巩固长期记忆，同时遗忘不重要的信息。
- **原则性**:框架的设计植根于神经科学的联想记忆模型和成熟的信息论统计度量(PPMI)，使其不仅在工程上可行，在理论上也具有坚实的基础。

## 7.2. 扩展方向

EMGAS作为一个基础框架，为未来研究开辟了多个富有前景的方向：

- **关系感知的激活扩散**:当前模型中的联想边是无类型的，仅有权重。未来的版本可以探索通过简单的模式匹配或轻量级分类器，为边赋予简单的关系类型(例如，“同义”、“因果”、“对比”)。在激活扩散过程中，算法可以根据查询的意图，对不同类型的边赋予不同的传播偏好，从而实现更具语境感知能力的推理。
- **混合衰减模型**:虽然指数衰减模型简洁有效，但一些认知科学研究表明，幂律衰减(power-law decay)可能更准确地描述人类的长期记忆遗忘曲线<sup>24</sup>。可以探索一种混合模型，对新近形成的、尚未巩固的记忆使用指数衰减，而对那些经过多次强化、激活值较高的长期记忆则应用更缓慢的幂律衰减。
- **用户专属的记忆图谱**:在多用户应用场景中，可以为每个用户维护一个独立的EMG实例。这将使得AI能够构建真正个性化的长效记忆，捕捉每个用户独特的语言习惯、兴趣领域和对话历史中形成的特定联想模式，从而提供高度定制化的交互体验。
- **与LLM的深度融合**:当前框架中，EMG作为外部记忆，与LLM的交互停留在上下文注入层面。未来的研究可以探索更深度的融合，例如，让LLM直接参与图谱的构建(如动态生成话题节点)，或者在激活扩散过程中利用LLM对节点和边的相关性进行实时评估，从而引导激活能量的流向。

## 引用的著作

1. 2410.05779v3.pdf
2. Spreading activation - Wikipedia, 访问时间为 九月 22, 2025,  
[https://en.wikipedia.org/wiki/Spreading\\_activation](https://en.wikipedia.org/wiki/Spreading_activation)
3. A spreading activation theory of memory - ACT-R, 访问时间为 九月 22, 2025,  
<http://act-r.psy.cmu.edu/wordpress/wp-content/uploads/2012/12/66SATH.JRA.JVL.1983.pdf>
4. Pointwise mutual information - Wikipedia, 访问时间为 九月 22, 2025,  
[https://en.wikipedia.org/wiki/Pointwise\\_mutual\\_information](https://en.wikipedia.org/wiki/Pointwise_mutual_information)
5. Pointwise Mutual Information (PMI) - Brainforge, 访问时间为 九月 22, 2025,  
<https://www.brainforge.ai/glossary/pointwise-mutual-information-pmi>
6. Pointwise Mutual Information (PMI) - Empathy First Media, 访问时间为 九月 22, 2025, <https://empathyfirstmedia.com/pointwise-mutual-information-pmi/>

7. Streaming Pointwise Mutual Information - Denison University, 访问时间为 九月 22, 2025, <http://personal.denison.edu/~lalla/papers/pmi-nips09.pdf>
8. Streaming Pointwise Mutual Information - NIPS, 访问时间为 九月 22, 2025, <https://proceedings.neurips.cc/paper/2009/hash/185c29dc24325934ee377cfda20e414c-Abstract.html>
9. Using PMI to Rank and Filter Edges in Graphs of Words - ResearchGate, 访问时间为 九月 22, 2025, [https://www.researchgate.net/publication/355084235\\_Using\\_PMI\\_to\\_Rank\\_and\\_Filter\\_Edges\\_in\\_Graphs\\_of\\_Words](https://www.researchgate.net/publication/355084235_Using_PMI_to_Rank_and_Filter_Edges_in_Graphs_of_Words)
10. clipperhouse/jargon - NPM, 访问时间为 九月 22, 2025, <https://www.npmjs.com/package/@clipperhouse/jargon>
11. 6 Best NLP Libraries for Node.js and JavaScript - Kommunicate, 访问时间为 九月 22, 2025, <https://www.kommunicate.io/blog/nlp-libraries-node-javascript/>
12. How to do stemming and lemmatization? / WinkJS - Observable, 访问时间为 九月 22, 2025, <https://observablehq.com/@winkjs/how-to-do-stemming-and-lemmatization>
13. Exploring LangChain.js: A Powerful NLP Library for JavaScript - Medium, 访问时间为 九月 22, 2025, <https://medium.com/aimonks/exploring-langchain-js-a-powerful-nlp-library-for-javascript-83ce270cdf80>
14. Yet Another Keyword Extractor (YAKE!) - GitHub Pages, 访问时间为 九月 22, 2025, <https://liaad.github.io/yake/docs/--home>
15. The Expert's Guide to Keyword Extraction from Texts with Python and Spark NLP, 访问时间为 九月 22, 2025, <https://www.johnsnowlabs.com/the-experts-guide-to-keyword-extraction-from-texts-with-spark-nlp-and-python/>
16. Keyword Extraction Methods in NLP - GeeksforGeeks, 访问时间为 九月 22, 2025, <https://www.geeksforgeeks.org/nlp/keyword-extraction-methods-in-nlp/>
17. Keyword Extraction Methods from Documents in NLP - Analytics Vidhya, 访问时间为 九月 22, 2025, <https://www.analyticsvidhya.com/blog/2022/03/keyword-extraction-methods-from-documents-in-nlp/>
18. TextRank Keyword Extraction Algorithm Using Word Vector Clustering Based on Rough Data-Deduction - PubMed Central, 访问时间为 九月 22, 2025, <https://PMC.ncbi.nlm.nih.gov/articles/PMC8808205/>
19. Automated Keyword Extraction – TF-IDF, RAKE, and TextRank - tiernok, 访问时间为 九月 22, 2025, <https://www.tiernok.com/posts/automated-keyword-extraction-tf-idf-rake-and-trextrank.html>
20. Application of Spreading Activation Techniques in Information Retrieval - ResearchGate, 访问时间为 九月 22, 2025, [https://www.researchgate.net/publication/225833731\\_Application\\_of\\_Spreading\\_Activation\\_Techniques\\_in\\_Information\\_Retrieval](https://www.researchgate.net/publication/225833731_Application_of_Spreading_Activation_Techniques_in_Information_Retrieval)
21. Revisiting the Autocorrelation of Long Memory Time Series Models - MDPI, 访问时间为 九月 22, 2025, <https://www.mdpi.com/2227-7390/11/4/817>

22. A two-phase model of collective memory decay with a dynamical switching point - PMC, 访问时间为 九月 22, 2025,  
<https://pmc.ncbi.nlm.nih.gov/articles/PMC9744905/>
23. Pre-Processing Implementation and Exponential Memory Decay, 访问时间为 九月 22, 2025, <http://www.cs.columbia.edu/~jebara/htmlpapers/ARL/node29.html>
24. The effect of memory decay on predictions from changing categories - ACT-R, 访问时间为 九月 22, 2025,  
<http://act-r.psy.cmu.edu/wordpress/wp-content/uploads/2012/12/184effect%20of%20memory%20decay.pdf>
25. When to Redis? When to MongoDB? [closed] - Stack Overflow, 访问时间为 九月 22, 2025,  
<https://stackoverflow.com/questions/5400163/when-to-redis-when-to-mongodb>
26. Redis OSS vs MongoDB - Difference Between NoSQL Databases - AWS, 访问时间为 九月 22, 2025,  
<https://aws.amazon.com/compare/the-difference-between-redis-and-mongodb/>
27. Redis vs MongoDB - Baeldung, 访问时间为 九月 22, 2025,  
<https://www.baeldung.com/java-redis-mongodb>
28. MongoDB Vs. Redis Comparison: Pros And Cons, 访问时间为 九月 22, 2025,  
<https://www.mongodb.com/resources/compare/mongodb-vs-redis>