arXiv:2512.17108v1 [cs.LG] 18 Dec 2025

ATOM：通过模块化复用实现高效的端侧视频-语言流水线

Kunjal Panchal[1], **Saayan Mitra** [2], **Somdeb Sarkhel** [2], **Haoliang Wang** [2], **Ishita Dasgupta** [2], **Gang Wu** [2], **Hui Guan** [1]

[1]马萨诸塞大学阿默斯特分校 [2]Adobe Research

摘要

近年来，视频语言模型取得了长足进展，催生了视频检索、字幕生成和视频汇编等强大应用。然而，由于模型加载冗余且执行过程碎片化，要在移动设备上高效运行此类多阶段流水线仍然面临挑战。我们推出 ATOM，这是一种端侧系统，通过重构视频语言流水线，实现快速、高效的执行。ATOM 将一个十亿参数模型拆解为可复用模块，例如视觉编码器和语言解码器，并在字幕生成、推理和索引等子任务间复用这些模块。这种以复用为核心的设计消除了重复的模型加载，并支持并行执行，从而降低端到端延迟，同时不牺牲性能。在普通智能手机上，与不采用复用的基线相比，ATOM 的执行速度提升了 27–33%，而性能仅略有下降（检索任务中 Recall@1 下降 ≤2.3，字幕生成任务中 CIDEr 下降 ≤1.5）。这些结果表明，ATOM 是一种实用且可扩展的方法，可在边缘设备上高效实现视频语言理解。

1 引言

视频语言模型（VLM）流水线（Jo 等，2024；Yan 等，2025；Long 等，2025）正迅速成为新兴移动应用的引擎，例如设备端视频检索（Tahboub、Gadgil 和 Delp，2015年；Zhang 等，2021）和视频组装（Yang 等，2023年）。在这些应用中，用户通过自然语言提示查找或拼接视频片段。这类应用依赖多阶段流水线，通常包含三个核心子任务：视频编码、字幕生成，以及下游推理或索引。在本地运行这些流水线非常理想，因为这样既能保护隐私，也无需依赖云服务器基础设施。

然而，由于内存和延迟限制，在移动设备上部署视觉语言模型流水线面临着根本性挑战。在标准流水线中，视频编码、字幕生成和脚本生成等子任务由不同的模型分别处理（Yang 等，2023年；Xu 等，2023年），每个模型都需要独立的内存占用和处理时间。例如，即使应用了 8 位量化，

mPLUG2（Xu 等，2023b）是一种常用于视频转字幕生成的模型，需要约 5.6GB 的随机存取存储器。添加 Llama 3.2 等脚本生成模型，还会额外占用 1.4GB。

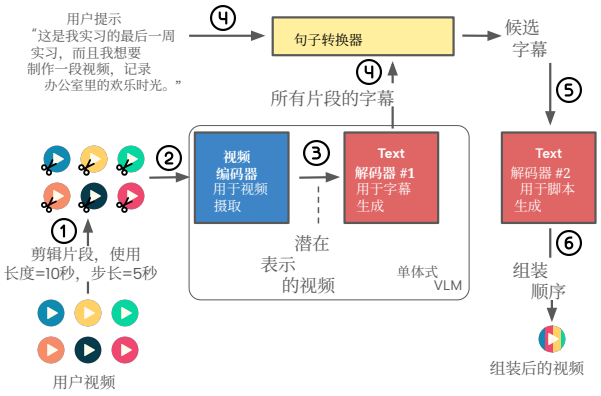
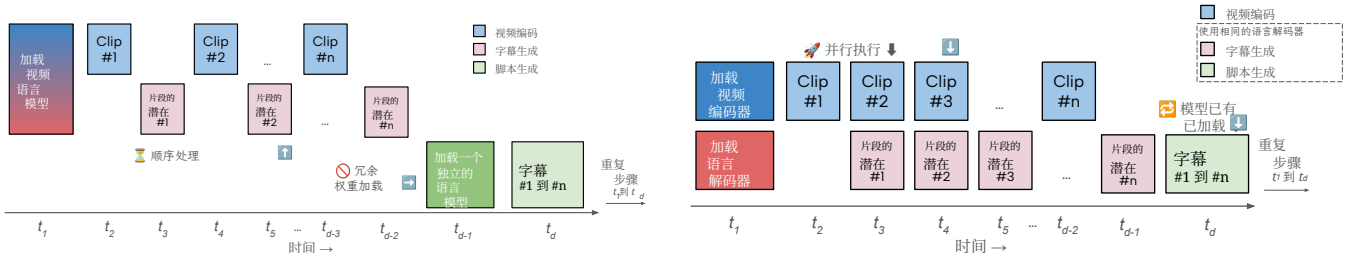


图1：用于视频组装的端到端设备端顺序流水线。给定 (i)用户视频和 (ii)用户提示，该流水线将所有视频片段转换为其文本表示，生成脚本，并据此组装选定的视频片段。

因此，运行完整流水线需要约 7GB 内存，这会给通常只有 6 到 8GB 随机存取存储器的移动设备带来负担。因此，如图1所示，流水线中的模型会在各阶段依次加载。这种连续加载阻碍了跨任务的流水线执行，并增加了端到端延迟。实际测试中，我们发现，在子任务之间重新加载这些模型的量化版本，延迟最多会增加 17%。这种低效不仅延长了等待时间，还限制了系统处理多个用户提示时的扩展能力，因为每个提示都会重新启动模型加载周期。

先前研究主要致力于优化单个模型的性能：加快词元生成速度，或降低每个模型的内存占用（Chu 等人，2023；Li 和 Li，2023；Xu 等人，2025）。尽管这些工作有效提升了流水线性能，却忽视了串联多个子任务时固有的流水线级低效问题 。在传统的视频组装流程中（图 2a），所有视频片段都会先经过视频编码器和字幕生成器，然后才调用独立的脚本生成器。这种串行执行意味着：（1）脚本生成仍在进行时，共享的视觉语言模型无法处理下一批视频片段；（2）使用独立的脚本生成模型会造成内存权衡：要么让两个模型同时驻留内存（这可能超出设备限制），要么承担依次加载它们的开销。



(a) 事实上的流水线中的顺序执行。子任务依次执行，每个阶段都会加载单独的模型（例如，在 t_1 和 t_{d-1}），从而导致更高的延迟和内存开销。

(b) ATOM 中的模块化、以复用为中心且支持并行执行的设计。通过将视频编码器和语言解码器模块化，并在不同输入之间并行执行二者（例如从 t_3 开始），该设计降低了流水线延迟。

图 2：顺序式（左）与 ATOM 的模块化并行（右）流水线执行。

这些限制加剧了延迟，并降低了边缘设备的响应能力。要克服这些问题，需要采用一种以复用为中心的模块化执行模型，以支持不同输入和子任务之间的并行处理。

我们的方法。 我们提出了 ATOM，这是一种以复用为中心的推理系统，通过解决流水线级低效问题来加速视频语言流水线。不同于为流水线中的每个子任务使用独立模型的顺序式基线，我们的方法采用单一的模块化模型架构来支持所有子任务，从而实现端到端复用，同时不影响模型质量。具体而言，ATOM 将 VLM 分解为持久且可复用的模块：视频编码器和通用语言解码器，这两个模块在整个应用生命周期内始终驻留在内存中。通过在不同子任务之间复用这些模块，并避免重复加载模型，ATOM 消除了数据搬移开销，并实现了并行执行，同时不会增加内存使用量。这种以复用为中心的设计与现有的模型压缩技术兼容，为进一步降低内存占用提供了更多机会。

图 2 对比了这种以复用为中心的设计与事实上的多模型基线。在当前做法中（图 2a），每个视频片段都会触发一个严格串行的工作流：设备首先加载一个整体式编码器—解码器对，依次为每个视频片段生成字幕，随后再加载一个独立的语言模型来生成脚本。我们的方法（图 2b）将视觉语言模型拆分为可单独调用、并发运行的编码器和解码器模块：解码器为片段 #1 生成字幕时，编码器已经可以处理片段 #2。随后，同一个解码器会处理所有字幕，为 videoassembly 生成脚本。这种模块化并行性在不增加内存占用的情况下，减少了模型加载时间的累积开销，并提高了硬件利用率，从而显著加快移动设备上的端到端处理速度。

实证结果。 我们在商用移动设备（包括 Google Pixel 5a、Pixel 8a 和 Samsung Galaxy S23）上，对 ATOM 在两项代表性视频语言任务（视频检索和视频组装）中的表现进行了评估。与传统的多模型流水线相比，ATOM：(a) 将模型加载延迟最多降低 50%；(b) 通过复用和并行处理，将端到端延迟最多降低 33%；(c) 在输出质量方面保持竞争力，性能指标仅有微小差异（例如，检索任务的 Recall @ 下降 1.3–2.3，字幕生成的 CIDEr 下降 0.6–1.5）；(d) 峰值内存几乎保持不变（仅比基线高 0.76% $\approx$ 40 MB），同时采用了更强大且可复用的文本解码器，仍能满足 6GB 智能手机内存预算。虽然我们的评估聚焦于视频检索和视频组装，但 ATOM 引入的以复用为核心的流水线设计广泛适用于各种视频语言应用，包括摘要生成、问答和时间事件定位。

本工作的贡献如下：

- 我们提出一种以复用为中心的流水线设计，为高效的视频语言推理指明了新的系统级方向。该设计通过在多个子任务间共享单个模型的内部模块，加速视频语言推理。这消除了执行期间管理多个模型所带来的开销。
- 我们开发了 ATOM，这是一种移动端推理系统，能够在不依赖服务器基础设施的情况下，在设备端执行完整的视频语言流水线。
- 我们在真实世界的智能手机上对 ATOM 进行了评估，结果表明，在保持各项任务高输出质量的同时，它显著降低了延迟，并提升了系统级效率。

2 相关工作

我们将相关工作分为两类：（1）视觉语言模型（VLM）的端侧推理及其部署策略；（2）面向视频中心任务（如组装和检索）的流水线。ATOM 通过以复用为中心的设计连接了这两个领域，从而降低了移动设备上视频语言流水线的端到端延迟。

设备端视觉语言模型。 近期研究探索了如何使视觉语言模型适配移动硬件。MOBILEVLM（Chu 等人，2023）和 SPOTLIGHT（Li 和 Li，2023）表明，通过将轻量级视觉编码器与语言解码器配对，十亿参数视觉语言模型能够在智能手机上高效运行。然而，这些工作主要针对基于图像的任务，并假设采用单阶段推理。

它们并未解决执行多阶段流水线所面临的挑战：反复加载模型且缺乏复用会带来显著的延迟开销。ATOM 与这些工作互为补充，着重于流水线内的架构复用，以加速设备端全栈视频语言推理，而非孤立地优化单个模型。

以视频为中心的流水线。视频语言任务（例如视频组装和视频检索）通常依赖由多个独立模型组成的模块化流水线，分别用于视频编码、字幕生成和推理（Yang 等，2023年；Xu 等，2023年c）。尽管这种方法效果良好，但由于需要针对每个子任务反复加载并运行不同模型，而移动设备的内存有限，因此会带来较高的运行时间开销。对于视频组装，RATV（Yang 等，2023年）和 TV-MGI（Yin 等，2024年）等方法利用强大的服务器托管模型，将视频片段与人工编写的脚本进行对齐。这些方法假设输入由专家编写，并且运行在资源充足的环境中。相比之下，ATOM 使用统一模型，直接在设备端自动完成脚本生成和片段选择，使这一过程对终端用户更加友好。在视频检索方面，CLIP4CLIP（Luo 等，2021）和 FROZEN（Bain 等，2021）等既有工作依赖视频与文本之间基于嵌入的对齐。较新的方法（Xu 等，2023年c；Jeong 等，2025年）集成基于大语言模型的推理，以提升检索质量，但依赖大型专有模型和云基础设施。这些方法无法部署在资源受限或隐私敏感环境中。ATOM 表明，视频检索、视频组装和字幕生成都可以在移动设备上由单一可复用模型支持，从而无需基于云的处理或人工监督。

3 ATOM 的设计

ATOM 通过以下三项关键设计原则消除流水线级冗余，从而加速视频语言流水线推理：

- 模块化：将视觉语言模型拆分为视频编码器和性能更强、可复用的语言解码器；两者均可独立调用，以便复用并发执行。
- 复用：模块常驻内存，并在各子任务之间共享，避免重复加载。
- 并行性：解耦的模块支持线程安全的并行执行，从而提高吞吐量和设备利用率。

3.1 视觉语言模型流水线的模块化执行

在传统的单体式视觉语言模型流水线中（Yang 等，2023年），视频字幕生成和脚本生成等任务（如图1所示）采用独立模型或紧密耦合的端到端模型实现。这会因串行执行和反复重新加载模型而导致较高的延迟。此外，单体式视觉语言模型会阻止不同子任务并发执行。

ATOM 通过模块化分解解决了这些低效问题。视觉语言模型（VLM）的视觉编码器和文本解码器组件被开放为可独立调用的模块。这样便能并行执行任务，例如在解码先前片段的字幕时，同时编码新的视频片段。模块化的另一项好处是，所有文本中心阶段都可以复用同一个高容量语言解码器。用一个共享解码器替代多个专用的轻量解码器：（i）消除了重复的模型加载；（ii）通过在所有文本中心子任务（例如字幕生成、检索索引和脚本生成）中应用更强的语言先验，促进了泛化；（iii）关键的是，使整个流水线能够在严格的移动设备内存预算内运行。相比之下，同时加载用于字幕生成和脚本生成的独立模型的流水线，通常会超出常见智能手机的 6GB 内存限制。复用单个高容量解码器避免了这一限制，同时还能提升质量。实证对比结果见第 5.2 节。

3.2 持久模块复用

即使采用了模块化，依赖任务专用模型的顺序实现仍可能在多阶段流水线的每个阶段重新加载不同模块，浪费计算资源和内存带宽。ATOM 通过持久复用策略避免了这类开销：将视频编码器和更强大的统一语言解码器等关键模块持续保留在内存中，供不同任务使用。例如，用于字幕生成的同一个解码器也会复用于脚本编写，从而无需在后续阶段加载单独的语言模型。

这种复用策略能够降低延迟，并避免在资源受限时发生内存抖动。如图 2a 所示，不复用的流水线会因反复加载和卸载模型而产生明显延迟，使用大型语言解码器时尤为如此。相比之下，ATOM（图 2b）会在整个执行过程中将模块保留在内存中，从而简化各子任务间的切换，并确保在移动硬件上运行更加顺畅。

尽管不同实现中的模型权重和激活值的内存占用相近，以复用为核心的设计降低了辅助内存用量（例如，权重加载期间的临时分配）。详细的内存细分见附录 F.4 和 F.5。

3.3 通过模块化设计实现并行性

将视觉语言模型模块化的一项关键优势是能够并行执行子任务，而这对于单体式或紧密耦合的模型来说是不可行的。传统的视频语言流水线由于模型不支持线程安全或内存占用过高，而必须串行执行：字幕生成必须等待视频编码完成，脚本生成则必须等待字幕生成结束。这会导致计算资源利用不足，并增加端到端延迟。

相比之下，ATOM 将解耦的编码器和解码器作为可独立调用、线程安全的模块暴露出来，并使其在整个执行过程中常驻内存。这种持久化的模块化设计使流水线各阶段能够重叠执行：例如，解码器为片段 #1 生成字幕时，编码器可以开始处理片段 #2。图 2b 展示了重叠的计算窗口如何提升吞吐量，并改善边缘设备上的硬件利用率。

虽然这种并行执行会增加峰值激活内存，但我们在附录 F.4 中表明，ATOM 通过消除重复加载模型所产生的临时内存开销抵消了这一影响，使总体内存用量仅增加 40MB。

对其他任务的适用性。 尽管为了具体说明，我们的分析聚焦于视频检索和组装，但 ATOM 中模块化、以复用为中心的设计也适用于其他视频语言任务，包括摘要生成和问答。我们通过附录 B 中描述的其他任务实现方案纲要对此进行了展示。

4 ATOM 的实现

我们基于视觉语言基础模型 mPLUG2（Xu 等，2023b）实现 ATOM。我们对其组件进行模块化，以支持持久模块复用和并行执行，从而实现高效的端侧推理。

我们选择 mPLUG2 有三个原因：（a）它在字幕生成和检索等视频语言任务中展现出强劲的性能（Wu 等，2024；Chen 等，2024；Wang 等，2024）；（b）它是开源且模块化的，便于有针对性地修改架构；（c）它兼容 TorchAO（torchao 2024）等量化和序列化框架，可用于移动端部署。原始 mPLUG2 由基于 BERT 的文本编码器和解码器（各含 0.3B 个参数）、CLIP-ViT/L-14 视觉编码器（0.4B 个参数）以及融合模块（0.5B 个参数）组成，参数总量为 1.5B。然而，它并未针对任务级复用进行优化：对于不同的语言任务（例如字幕生成与脚本生成），解码器都需要单独微调，这会增加训练和运行时开销。

为实现跨子任务复用，我们进行了两项关键修改。首先，舍弃文本编码器和融合模块，因为我们的任务不需要它们。其次，将基于 BERT 的解码器替换为基于 1B 参数 Llama 3.2 的自回归解码器（Grattafiori 等，2024年）。该解码器可在字幕生成和推理任务之间实现泛化，使同一模块能够承担两种角色，而无需重新训练。

由此得到的模型称为 mPLUG2+，由基于 CLIP 的视频编码器和基于 Llama 的文本解码器组成，共包含 1.4B 个参数。我们在视频描述数据集上对模型进行微调，以对齐视觉和文本组件；脚本生成无需额外训练。与原始 mPLUG2 相比，此次更新提升了字幕生成性能（CIDEr 提高 2.8–3.7）和检索性能（Recall@1 提高 3.8–4.9），如第 5.2 节所示。为支持以复用为中心的执行，每个模块都被独立序列化，并在流水线执行期间保持持久加载。这种设计既支持共享复用，也支持子任务间线程安全的并行性，从而在配备 6GB+随机存取存储器的移动设备上实现可靠执行。

对其他视觉语言模型的适应性。虽然我们的实现以 mPLUG2+ 为基础模型，但 ATOM 的复用中心设计广泛兼容许多采用模块化架构的现代视觉语言模型，例如 BLIP-2、Flamingo 和 VideoCoCa。这些模型将视频编码器与语言解码器解耦，因此适合持久复用。尽管早期采用紧密耦合架构、带有联合跨模态注意力层的模型可能难以进行这种分解，但在移动设备上部署通常需要模块化，以实现低延迟和高内存效率；这一假设也日益体现在较新的模型架构中。

量化。 为支持在内存受限的移动硬件上部署，我们采用模型量化来减少内存占用。我们使用 TorchAO 进行动态量化，以便在移动设备上高效部署视频语言模型。我们重点处理线性层和嵌入层（占内存使用量的 91% 以上），实现了 ARM CPU 上的 int8 兼容性和序列化，同时避免了不可靠的三维卷积量化。有关量化、算子兼容性、序列化和部署约束的详情，请参见附录 C 和 D。

5 结果与讨论

我们在两个具有代表性的视频语言流水线（视频组装和视频检索）上评估 ATOM，并展示以下关键发现：(a) 修改后的 mPLUG2+架构支持任务复用，且不会降低性能；(b) ATOM 以复用为核心的设计通过避免重复加载权重并实现并行处理，大幅降低了端到端延迟。

我们首先在第 5.1 节详细介绍实验设置，随后在第 5.2 和 5.3 节展示结果并进行分析。更多定性示例和结果见附录 E 和 F。

5.1 实验设置

任务与模型。 我们选择视频检索和视频组装，分别代表视频语言任务谱系的两端：基于检索的推理和生成式故事叙述。尽管两者有所不同，但都能从作为共享中间步骤的字幕生成中获益。这样一来，我们便能在不同任务间复用同一个语言解码器，减少模型加载阶段的延迟和内存开销。服务器端系统或许能够负担针对特定任务的模型，而我们的设计则侧重复用，以满足移动端限制。附录 B 讨论了这种方法如何扩展到摘要等其他任务。

两条流水线都依赖三个核心模块：用于视频编码的视觉编码器、用于字幕生成（以视频特征为条件）的语言解码器，以及用于将用户提示与生成的字幕进行匹配的句子转换器。语言解码器还会复用于脚本生成，从而体现 ATOM 以复用为核心的设计所带来的效率。

如第 4 节所述，我们使用 CLIP-ViT/L-14（0.4B 个参数）作为视觉编码器，并使用 Llama 3.2（1B 个参数）作为语言解码器。该模型使用四块 NVIDIA A100 GPU 和下文所述的数据集进行视频字幕生成微调。对于语义匹配，我们使用 all-MiniLM-L6-v2（Reimers 和 Gurevych 2019），这是一种包含 2,270 万个参数的句子转换器。

数据集。 我们使用三个数据集对改进后的 mPLUG2+ 进行微调，并评估其在视频字幕生成、视频

检索和定性视频组装方面的表现（附录 E）：(a) MSR-VTT（Xu 等，2016年）、(b) MSVD（Chen 和 Dolan，2011年）以及 (c) DiDeMo（Hendricks 等，2018年）。模型使用这三个数据集各自训练集的合并数据进行训练，并在各自的测试集上进行评估。前两个数据集也用于训练原始 mPLUG2。

MSR-VTT包含10K个YouTube视频和200K条字幕，分为6.5K个训练样本、0.5K个验证样本和3K个测试样本。MSVD包含1,970个带字幕的YouTube片段，分为1.2K/100/670个训练/验证/测试样本。DiDeMo包含10K个Flickr视频，分为8K/1K/1K个样本，每个视频配有四条文本描述。

对比方法。先前的视频组装研究（Yang 等，2023；Yin 等，2024）依赖人工编写的脚本来检索相关片段。相比之下，ATOM 根据用户提示从视频片段生成脚本。这些先前系统也都是闭源的，因此无法进行可复现的比较。在视频检索方面，现有方法主要关注（a）架构改进（Xu 等，2023年；Xu 等，2023年）或（b）提高检索相关性（Tevissen、Guetari 和 Petitpont，2024年），但面向移动端环境设计的方法（Zhang 等，2021；Tahboub、Gadgil 和 Delp，2015年）并未利用大语言模型进行推理。

由于这些限制，我们使用侧重于性能和效率的系统级基线来评估 ATOM：（a）一台 96 核 x86 64 CPU服务器，分别采用和不采用 ATOM 的复用中心设计；（b）同一台配备 NVIDIA A40 GPU 的服务器，分别在量化和全精度模式下进行评估，且同样分别采用和不采用复用；（c）在移动设备上运行不使用复用的 ATOM，每个子任务都单独加载和卸载模型。第三种基线体现了缺乏协调的典型多阶段流水线。与之比较可以单独衡量 ATOM 的模块化复用在延迟和内存效率方面带来的收益。

指标。 为了评估**视频字幕生成子任务**的性能，我们采用四项指标：（a）BLEU@4，（b）ROUGE-L，（c）METEOR，以及（d）CIDEr。BLEU@4 计算生成字幕与参考描述之间最多 4-gram 的 n-gram 精确率几何平均数，并应用简短惩罚。ROUGE-L 通过查找最长公共子序列来评估内容重合度和词序。METEOR 计算精确率和召回率的加权调和平均数，并纳入多种语言学特征。CIDEr 使用 TF-IDF 加权的 n-gram 余弦相似度来衡量相似性，突出具有辨识度的短语。这些指标综合评估字幕质量，考量精确率、召回率、语义相似度和独特性等因素。

为了评估**视频检索任务**性能，我们使用 (a) Recall@1、(b) Recall@5 和 (c) Recall@10。Recall@k衡量在视频检索流水线的前 k 个结果中成功检索到的相关视频所占比例。

与此同时，**视频组装**的评估涉及更多人为因素，而且尚无适用于该任务的既定指标。因此，我们按照附录 E 所示，对 ATOM 进行了定性分析，以衡量其效能。对于通过句子转换器进行的**语义匹配**任务，皮尔逊相关系数用于衡量两个句子嵌入表示之间的线性关系，并根据它们数值的协变程度量化语义相似度。

延迟以秒为单位。我们测量了加载模型、运行流水线的各个组件或子任务（视频编码、字幕生成），以及运行端到端流水线的延迟。**内存消耗**以吉字节 (GB) 为单位。

硬件。我们在 3 台计算能力各异的实体移动设备上评估了 ATOM：(a) Google Pixel 5a（6GB 随机存取存储器）、(b) Google Pixel 8a（8GB 随机存取存储器）和 (c) Samsung Galaxy S23（8GB 随机存取存储器）。我们使用 PyTorch Mobile 作为推理引擎。我们还使用了专为 PyTorch Mobile 构建的 TorchAO，通过算子融合、量化和轻量级运行时自适应来优化执行。鉴于 TorchAO 与 GPU 和 NPU 等其他处理单元的兼容性有限（其他研究也观察到了这一限制（Xu 等人，2025年，2023年）），实验重点是在移动 CPU 上执行端到端任务流水线。不过，ATOM 仍可适配其他处理单元，因此与硬件加速器的选择正交。

超参数。我们采用训练 mPLUG2 时使用的超参数（Xu 等，2023b）。视频描述任务的训练持续 10 轮，学习率为 2e-5。批大小设为 128。字幕生成的最大长度设为 23 个词元，这是固定 JIT 跟踪解码循环迭代次数所需的设置。字幕生成采用 4 的束大小。视频经过预处理，采样率设为每秒 6 帧，每帧的分辨率降低至 224×224。附录 G 讨论了超参数选择。

5.2 性能

本节将展示，在设备端视频语言流水线中集成更强大的语言解码器，对检索和字幕生成任务带来的优势。由 LLaMa 文本解码器驱动的增强版 mPLUG2+，与采用 BERT 解码器的原始 mPLUG2 相比，预测性能显著提升：未量化时 CIDEr 提高 2.8–3.7 分，量化后提高 6.8–7.2 分。此外，量化对性能影响极小，仅使 CIDEr 降低 0.6–1.3 分，同时支持在移动端和边缘设备上高效部署。这些进展使我们得以构建首个完全可运行的设备端视频组装流水线，让实时视频理解成为可能。

我们首先讨论视频字幕生成和视频检索任务的性能，具体内容分别见表1和表2。由于缺乏视频组装任务的定量指标，我们邀请了10位人类参与者对组装视频进行了定性分析，结果见附录 E。

与原始 mPLUG2 相比，我们修改后的 mPLUG+ 使 BLEU@4 提高了 1.9–5.5 分，表明生成的字幕与参考字幕之间的对齐效果更好。ROUGE-L 也提高了 2.3–2.8 分，反映出短语匹配准确率有所提升；METEOR 提高了 2.1–2.6 分，体现出更好的

模型变体	MSR-VTT				MSVD			
	B	R	M	C	B	R	M	C
	未量化（32 位权重）							
mPLUG2	56.9	34.0	68.2	79.4	73.4	46.8	84.7	163.6
mPLUG2+	62.4	36.8	70.8	83.1	75.3	49.1	86.8	166.4
	采用量化（8 位权重）							
mPLUG2	52.4	30.8	65.4	75.3	70.8	43.2	80.7	158.1
mPLUG2+	60.7	35.9	68.6	82.5	74.2	46.3	84.5	164.9

表 1：**原始 mPLUG2（使用 BERT 文本解码器）与 mPLUG（使用 LLaMa 文本解码器）在量化和全精度版本下执行视频字幕生成**任务的性能比较。所有 4 项指标 B（BLEU@4）、R（ROUGE-L）、M（METEOR）和 C（CIDEr）均为“越高越好”。与使用较弱文本解码器的原始 mPLUG 相比，我们的 ATOM 方法 mPLUG2+即使经过量化，性能下降幅度也更小。

模型变体	MSR-VTT			MSVD		
	R@1	R@5	R@10	R@1	R@5	R@10
未量化（32位浮点权重）						
mPLUG2	52.6	75.8	83.1	65.3	82.0	90.6
mPLUG2+	57.2	80.7	85.1	69.1	84.5	93.8
采用量化（8位整数权重）						
mPLUG2	48.7	72.6	80.5	62.1	77.3	87.9
mPLUG2+	55.8	78.4	82.9	66.8	82.7	90.3

表 2：原始 mPLUG2（采用 BERT 文本解码器）与我们修改后的 mPLUG2+（采用 Llama 文本解码器）在视频检索任务中的性能比较，涵盖量化和全精度版本。三个指标 R@1（Recall@1）、R@5（Recall@5）和 R@10（Recall@10）均为“越高越好”。与采用较弱文本解码器的原始 mPLUG 相比，即使经过量化，ATOM 的 mPLUG2+ 性能下降幅度也更小。

词对齐。此外，衡量描述信息量的 CIDEr 分数提升了 2.8–3.7 分。即使经过量化，mPLUG2+ 也始终比原始模型高出 6.8–7.2 个 CIDEr 分数。与全精度 mPLUG+ 相比，量化导致 CIDEr 分数略微下降了 0.6–1.5 分，但定性差异可以忽略不计，详见附录 E。

同样，表 2 中的检索结果显示，我们的方法在 MSR-VTT 和 MSVD 数据集上均取得了稳定的性能提升。DiDeMo 数据集上的扩展结果见附录 F.6。全精度 mPLUG2+模型的 Recall@1 比 mPLUG 高出 3.8–4.9 分，表明其检索最相关视频的能力更强。Recall@5 和 Recall@10 也呈现出类似趋势，进一步证明了我们的模型在不同排名位置上的稳健性。即使经过量化，mPLUG2+ 仍比原始 mPLUG2 具有检索优势，Recall@1 仅下降 1.3–2.3 分，同时仍比量化后的 mPLUG 模型高出 4.7–7.1 分。此外，我们在附录 F.3 中表明，用于检索和组装流程中描述到提示的匹配的 all-MiniLM-L6-v2 句子转换器，对系统的端到端质量影响可以忽略不计，即使经过量化也是如此。这确保了系统在资源受限环境中仍能高效、准确地进行匹配。

5.3 延迟

本节介绍 ATOM 以复用为中心的设计如何通过高效的流水线执行，使视频组装和视频检索任务的延迟降低 27–33%。

表3显示，ATOM 在资源受限的设备上显著降低了延迟。在 Google Pixel 5a 上，我们以复用为中心的设计将端到端视频检索延迟降低了 33.06%，并将视频组装延迟降低了 30.78%。Google Pixel 8a 和 Samsung Galaxy S23 上也呈现出类似趋势：检索延迟分别降低约 30.43% 和 29.14%，组装延迟分别降低 27.69% 和 27.54%。这些延迟降低得益于计算模块的并行执行与摊销复用，从而显著减少了冗余处理。

这些改进背后的关键因素，是 ATOM 能够避免重复的模型加载和执行开销。传统的顺序式多模型实现会针对不同任务重新加载模型，带来显著的延迟惩罚，尤其是在计算能力受限的移动设备上。通过复用大型视频语言模型的模块，具体来说就是我们修改后的 mPLUG2+，ATOM 消除了冗余计算，并让不同流水线阶段能够并发执行，从而更有效地利用硬件并行性。这使视频组装和视频检索的延迟显著降低；如表格中“（不复用）”对应的结果所示，若采用顺序执行，则会造成严重瓶颈。

另一个关键见解是对高端服务器配置与移动硬件的比较。在高端配置中，GPU（例如 NVIDIA A40）能够通过高效的向量化计算大幅降低延迟，而移动设备则缺乏此类加速能力。在这些环境中，优化执行流程尤为重要。结果证实，ATOM 注重复用的方法有效弥合了这一性能差距，无需额外的专用硬件即可实现超过 30% 的端到端延迟降低。

此外，量化对延迟的影响进一步凸显了 ATOM 设计的重要性。虽然量化通常会降低模型的内存占用，但它会引入动态转换开销，从而增加未采用复用的视频流水线实现中的延迟。然而，ATOM 通过并行执行视频编码器和语言解码器来缓解这一问题，避免性能下降。在资源受限的移动设备配置中，这些改进尤为明显；在此类环境下，我们的方法能够防止量化操作按顺序处理所造成的过度延迟。

			分组件*				端到端	
	平台变体	模型加载	视频编码 (CLIP-ViT)	字幕生成 (Llama)	索引 (MiniLM)	脚本生成 (ST +解码器)	视频检索	视频组装
高端硬件	未量化							
	x86_64 CPU（不复用）	6.19	1.68	1.15	10.35	11.09	36.05	40.67
	x86_64 CPU +GPU（不复用）	8.66	0.15	0.08	6.48	6.84	12.12	14.95
	x86_64 CPU (ATOM)	3.44 (↓ 44.43%)	1.68	1.15	10.35	11.09	26.58 (↓ 26.27%)	30.28 (↓ 25.55%)
	x86_64 CPU +GPU (ATOM)	4.38（↓ 49.42%）	0.15	0.08	6.48	6.84	8.65（↓ 28.63%）	10.46（↓ 30.03%）
	采用量化							
	x86_64 CPU（不复用）	7.32	2.95	2.14	13.66	15.40	44.84	47.59
	x86_64 CPU +GPU（不复用）	8.89	0.46	0.32	10.78	13.02	19.20	22.18
	x86_64 CPU (ATOM)	3.64（↓ 50.27%）	2.95	2.14	13.66	15.40	32.12（↓ 28.37%）	34.69（↓ 27.11%）
	x86_64 CPU +GPU（ATOM）	4.56（↓ 48.71%）	0.46	0.32	10.78	13.02	14.02 (↓ 26.98%)	16.38 (↓ 26.15%)
资源受限-受限硬件	Google Pixel 5a（无复用）	32.94	8.68	5.57	24.39	26.61	162.33	167.52
	Google Pixel 8a（不复用）	29.63	6.34	4.22	21.20	22.41	147.04	153.59
	Samsung Galaxy S23（不复用）	28.43	6.83	3.54	21.03	22.68	146.47	152.35
	Google Pixel 5a（ATOM）	17.23（↓ 47.69%）	8.68	5.57	24.39	26.61	108.66（↓ 33.06%）	115.95（↓ 30.78%）
	Google Pixel 8a（ATOM）	15.87（↓ 46.44%）	6.34	4.22	21.20	22.41	102.30（↓ 30.43%）	111.06（↓ 27.69%）
	Samsung Galaxy S23（ATOM）	14.12（↓ 50.33%）	6.83	3.54	21.03	22.68	103.78 (↓ 29.14%)	110.39 (↓ 27.54%)

表3：**延迟（单位：秒，越低越好）**：ATOM 与顺序（不复用）实现的对比。括号中的数字（↓）表示 ATOM 相较于顺序基线节省的百分比。该表报告了高端服务器和资源受限的移动设备上各个子任务（视频编码、字幕生成、索引和脚本生成）的**组件级延迟**，以及检索和视频组装任务的**端到端延迟**。GPU：NVIDIA A40。***注**：组件级延迟按每个输入分别独立测量。然而，端到端延迟反映的是对 5 个输入的执行情况，展示了 ATOM 并行执行和模型复用摊销带来的优势。这种设计通过仅加载一次模型模块，并支持跨输入并发处理，显著缩短了整体运行时间，使端到端延迟低于各组件时间的朴素总和。

总体而言，这些研究结果验证了 ATOM 以复用为中心的流水线执行的有效性。通过策略性地利用模块化复用和并发执行，ATOM 显著加速了视频检索和组装任务，使实时视频处理即使在移动硬件上也成为可能。这些结果凸显了优化计算复用对于实现低延迟、高效率人工智能驱动的应用的重要性。

5.4 附加结果

为进一步证明 ATOM 的实用性，附录 F 提供了有关延迟-性能权衡、存储可行性和内存效率的结果。

(a) 附录 F.1 中在 Google Pixel 5a 上开展的消融实验展示了不同执行配置下延迟与性能之间的权衡，这些配置包括视频分辨率、字幕长度、帧率、束大小和批大小。研究人员可据此根据具体的部署约束调整系统。

(b) 我们还在附录 F.2 中分析了模型存储可行性。模块化视频语言模型（包括 SentenceTransformer）总共占用 ∼1.3GB，因此适合部署在配备 6–8GB 随机存取存储器的常见智能手机上。

(c) 最后，附录 F.4 和 F.5 详细细分了 ATOM 相较于不采用复用方案的变体的内存占用，并讨论了延迟-内存权衡。由于 ATOM 在编码器和解码器并行运行时会将二者保留在内存中，因此它们运行时，激活占用增加了 182MB，但这被消除了重复模型加载过程中产生的临时缓冲区所抵消。最终，峰值 RAM 仅增加了 0.76%（≈ 40MB），影响微乎其微，使完整流水线仍远低于普通智能手机的 6GB 预算。

6 结论

在这项工作中，我们提出了 ATOM，这是一种高效的设备端系统，能够在普通移动设备上实现端到端视频语言流水线。ATOM 采用以复用为中心的设计，将单个视觉语言模型的组件模块化并重新用于流水线的各个阶段。这种方法不仅解决了模型加载延迟这一关键挑战，还确保系统能够实际部署在随机存取存储器仅为 6GB 的设备上。我们在商用智能手机上进行的评估表明，与缺乏模块化或复用、事实上采用顺序执行的多模型流水线相比，ATOM 在下游任务上实现了更强的性能（视频组装的 CIDEr 分数最高提升至 3.7，检索的 Recall@1 达到 4.9），且量化带来的性能下降极小，同时显著提升了运行效率（最高达 33% ）。定性评估也印证了 ATOM 的定量增益。通过免去对外部服务器的依赖，ATOM 提升了隐私保护、降低了基础设施成本，并使先进的视频语言应用即使在带宽受限或离线环境中也能使用。

A 用例：视频组装

我们详细介绍了 ATOM 如何完全在设备端执行视频组装任务，展示其模块化、以复用为中心的设计。此示例突显了流水线架构，以及视频编码器模块与语言解码器模块之间的交互。

A.1 流水线概述

给定用户提示（例如“制作一个精彩集锦”）和一组用户提供的视频，流水线按以下步骤运行：

1. **视频分段**。输入视频会被划分为固定时长（$\ell = 10s$）且相互重叠的片段，步幅为$s = 5s$，从而实现更精细的时间粒度，并减少每次推理调用的内存负载。2. **视频字幕生成**。每个片段都会通过视觉语言模型进行两阶段处理：

- 视频编码器提取高维视觉嵌入表示。

• 文本解码器根据视觉嵌入生成简洁的自然语言描述。3. **嵌入与排序**。字幕和用户提示通过句子转换器进行嵌入。通过计算字幕嵌入与提示嵌入之间的余弦相似度，得出片段的相关性。4. **脚本生成**。使用同一视觉语言模型的文本解码器（此时以纯文本模式运行），将相关性最高的前k条字幕编排成连贯的视频脚本。该脚本决定所选片段的顺序和旁白。5. **最终组装**。在设备上将选定的片段拼接起来，生成最终视频输出。

A.2 面向移动设备约束的预处理

为降低推理延迟和内存开销，我们采用以下策略预处理视频片段：

• **降低分辨率和帧率**。 我们对智能手机视频输入进行空间和时间维度的下采样。

• **片段格式化**。 每个视频片段都会重塑为大小为 (1,3, 6, 224,224) 的张量，分别对应批大小、通道数、采样帧、高度和宽度。此尺寸在场景级理解的准确性与效率之间实现了良好的权衡。

从分割到脚本生成的整个流水线均完全在设备端运行，并通过持久模块复用避免不必要的重新加载。主论文中的图1展示了这一过程。

B 对视频语言任务的更广泛适用性

虽然视频组装是我们的主要案例研究，但 ATOM 以复用为核心的架构自然适用于广泛的视频语言任务，包括视频检索、摘要生成、问答等。这些应用具有共同的多阶段结构：需要先进行视频编码，再进行基于语言的推理或匹配，因此非常适合 ATOM 模块化、持久化且支持并行执行的设计。

在视频检索中，每个片段首先由视觉编码器处理，以生成嵌入表示。ATOM 不会存储高维潜在表示（例如 2048×1024，每个片段最多可能占用 8MB），而是使用语言解码器生成紧凑且语义丰富的文本描述。这种基于文本的表示将每个片段的存储开销降至仅几百字节，并可借助轻量级句子转换器模型（例如 all-MiniLM-L6-v2， ∼90MB）快速进行基于相似度的匹配。以复用为中心的流水线会在所有检索步骤中复用视觉编码器和语言解码器，从而避免重复加载模型，并显著降低延迟和内存使用量。因此，即使在内存受限的移动设备上，也能实现准确的视频检索。

在视频摘要生成中，模型首先使用编码器—解码器组合，为较短的视频片段生成场景级字幕。随后，这些中间字幕会被送回同一个语言解码器，以生成更高层次的摘要。由于解码器在初始字幕生成阶段加载后会继续保留在内存中，摘要生成阶段无需额外加载模型，从而提高效率，并确保各阶段的语义表示保持一致。这种复用不仅降低了开销，还通过在所有阶段使用同一个高容量解码器，保障了输出质量的一致性。

对于基于视频的问答（VideoQA），系统会再次对视频进行分段，并使用视觉编码器和解码器生成字幕。给定用户问题后，解码器会基于生成的字幕或原始视觉嵌入进行推理，从而生成答案。由于解码器在前序阶段已驻留内存，因此可直接复用于推理，无需加载任何问答专用模型，从而降低延迟和内存占用。

在所有这些应用中，ATOM 的主要优势在于其系统级复用策略。通过让已加载的模块持续驻留内存，ATOM 无需在每个子任务边界处加载和卸载模型。这既能持续降低运行时间和内存占用，也能简化软件流水线。此外，ATOM 在字幕生成、摘要生成、检索和推理等任务中复用同一个语言解码器，确保输出之间的语义一致性，并避免使用不同专用模型可能导致的偏差。

这一设计的通用性支撑着更广阔的愿景：针对一系列视频语言理解任务实现模块化统一执行，并针对设备端部署进行优化。因此，ATOM 不仅是一种适用于视频组装的移动友好型架构，也是面向未来边缘设备多模态应用的可复用骨干。

C 量化

为实现通用移动硬件上的部署，我们对 ATOM 中使用的视频语言模型组件应用动态量化。不同于以往仅关注缩小模型大小以适配内存的工作，我们采用量化的目标是在保持运行效率的同时

确保与移动后端兼容。这样可确保复用的模块能够持续驻留在内存中，并在不同任务间高效运行。

对视频语言模型进行量化面临着一些独特的挑战。首先，许多常用的量化方法不支持视频编码器中使用的关键算子，例如三维卷积。此外，一些库不支持 int8 量化，或缺乏基于 ARM 的移动部署所必需的序列化能力。最后，量化必须以尽可能降低对模型精度影响的方式进行。

为应对这些挑战，我们采用了基于 TORCHAO（torchao 2024年）的动态量化策略。我们仅针对线性层和嵌入层进行量化，这两者合计占模型内存占用的 91% 以上。TORCHAO 支持这些算子，并通过 int8 数据类型实现与 ARM CPU 执行的兼容性。它还支持序列化，使我们能够在各种安卓设备上打包和部署量化模型。

鉴于各量化库对三维卷积层的支持不够可靠，我们有意不对其进行量化。相反，我们通过预处理来降低视频编码的计算成本，包括在推理前对帧率和分辨率进行下采样。这种方法能够降低运行时间负载，同时不会过度牺牲场景理解的准确性。

在我们的评估中，量化仅带来了适度的性能下降：在图像描述任务上约下降 1.5 个 CIDEr 点，在视频检索上下降 2.3 个 Recall@1；同时，它使得仅配备 6GB 随机存取存储器的设备也能够实现完整的流水线部署。我们还观察到模型加载时间缩短、峰值内存使用量降低，从而进一步改善了端到端延迟。

值得强调的是，ATOM 中的量化补充了我们的核心设计目标：通过架构复用降低运行时开销。我们的贡献不在于开发新的量化方法，而在于确保动态量化等标准技术能够无缝集成到以复用为中心的执行模型中，从而在移动端限制下支持高容量模块持久地在内存中运行。

D 序列化和部署量化模型

在手机等资源受限的设备上，量化模型的存储和部署与高端服务器上的基于 Python 的执行存在显著差异；在高端服务器上，模型权重通常以字典形式存储。对于移动端部署，必须通过追踪量化模型完整的前向传播（包括跨多个模块的嵌套前向传播）来对其进行序列化。这种方法确保了与移动执行环境的兼容性，并支持在受限硬件上进行高效推理。

为实现量化视频语言模型的保存和加载，我们采用了 torch.jit.trace，将前向传播期间执行的操作记录到计算图中。不过，为确保兼容性，我们进行了几项重构：

1.**静态类型的前向传播**：PyTorch JIT（即时）编译器在变量赋值和条件操作期间难以进行动态数据类型推断。我们重构了子模块（视觉编码器和文本解码器）中的所有前向传播，使其使用静态类型。2. **依赖数据的控制流**：由于文本解码器无法处理动态输出长度，因此对其进行跟踪颇具挑战。作为一种变通方法，我们使用能生成最大字幕长度的样本输入，并从输出中删去多余的词元。

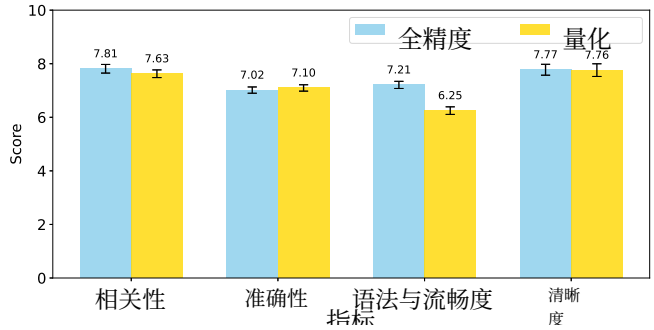
3.**替换 Torch 操作**：某些矩阵操作（例如 torch.rsub 和 torch.zeros）与 JIT 跟踪不兼容。我们将它们替换为等效操作，例如组合使用 torch.mul 和 torch.sub，或使用 torch.zeros_like。

完成前向传播的跟踪后，我们针对移动端部署应用了额外的优化，包括：(a) **算子融合**：我们合并了经常成对出现的操作所对应的内核，例如线性层和 ReLU，以减少执行开销并提高运行效率。(b) **参数提升**：为尽量减少结构元数据的存储，我们对模型进行了重构，将所有算子提升至前向传播的顶层，从而简化执行并减少总体内存占用。这些优化进一步提高了在资源受限的移动设备上部署量化视频语言模型的效率和可行性。

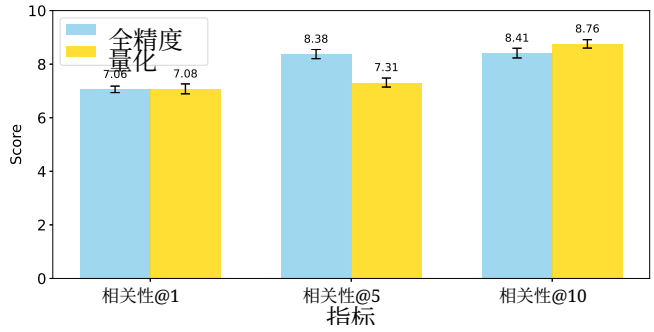
E 定性分析

我们邀请了 10 名人类参与者，评估 ATOM 修改后的 mPLUG2+ 在全精度和量化版本下执行视频字幕生成、视频检索和视频组装任务的表现。结果如图 3 所示。定性指标的评分范围为 0（对应指标得分最低）至 10（对应指标得分最高）。下文将讨论与这些评分相关的观察结果。

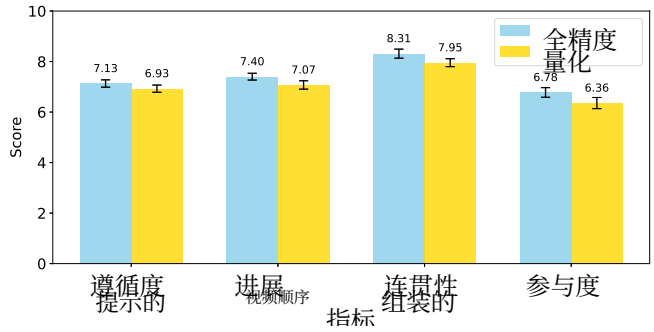
对于视频字幕生成任务，我们从以下方面评估生成的字幕：(a) 相关性：字幕是否符合视频的主题或主旨；(b) 准确性：字幕是否准确描述了视频中物体的细节；(c) 语法和流畅度：字幕在语言上的正确性与自然程度；(d) 清晰度：字幕是否含义明确。每位参与者都观看了 10 段视频，并查看了由全精度模型和量化模型生成的字幕。图 3a 展示了各项评估指标的平均分，误差线表示标准差。在四项指标中，参与者对全精度模型和量化模型的评分均值相差 0.1–1 分。两种模型的评分标准差均介于 0.7 至 1.2 之间，这表明观察到的均值差异在统计上并不显著。因此，结果显示，全精度模型与量化模型在字幕质量上的差异微乎其微；这与表 1 所示的定量结果一致。表 4 展示了少量生成字幕的示例，进一步印证了二者之间的差异微乎其微



（a）视频字幕生成



(b) 视频检索



(c) 视频组装

图 3：三项任务的定性分析：(a) 视频字幕生成，(b) 视频检索，(c) 视频组装。结果基于 10 位参与者。不同指标得分的量表为 $[0, 10]$。

全精度模型和量化模型的生成能力。

在视频检索任务的背景下，我们通过展示排名前-k 个视频，并询问视频内容与书面提示的契合程度，评估检索到的视频（给定提示）在前-k名的相关性（Relevance@k）。我们请每位参与者撰写 10 条提示，并评估全精度模型和量化模型检索到的视频在前 1、前 5 和前 10 名的相关性。图 3b 展示了上述指标的平均分，误差线表示标准差。所有参与者的前-k 名平均分在全精度模型和量化模型之间介于 0.10-1.52 之间。全精度模型的标准差介于 0.75-1.02 之间，而量化模型的标准差介于 0.92-1.13 之间。结果表明，全精度模型和量化模型的前-k 名平均分处于相近范围，且标准差存在重叠。这说明量化对视频检索性能的影响很小，因为两种模型的得分分布相近。

最后，对于视频组装任务，定性评估依据以下指标进行：(a) 提示遵循度：组装后的视频内容是否与给定提示相符；(b) 视频顺序：组装中的视频顺序是否自然；(c) 组装的连贯性：组装后的视频片段在风格和内容上是否协调；(d) 吸引力：组装后的视频是否有趣或引人入胜。每位参与者需撰写 10 条提示，并对全精度模型和量化模型组装的视频进行上述指标的评估。图 3c 展示了上述指标的平均得分，误差线表示标准差。全精度模型的平均得分介于 6.8 至 8.3 之间，量化模型的得分介于 6.3 至 7.9 之间。全精度模型的标准差介于 0.81 至 1.17 之间，量化模型的标准差介于 0.77 至 1.19 之间。标准差的重叠表明，量化对视频组装任务的影响也可以忽略不计，因为两种模型的得分分布相近。

因此，在视频字幕生成、视频检索和视频组装这三项任务中，定性评估表明，量化对性能的影响极小；全精度模型与量化模型的平均分数及标准差存在显著重叠。

F 附加结果

本节将介绍更多结果，以深入了解系统的效率和设计选择，包括消融实验、存储需求、量化效应以及内存占用分析。

F.1 消融实验

表5展示了在 Google Pixel 5a 上，改变输入配置和超参数对 ATOM 修改后的 mPLUG2+ 的延迟和性能的影响。默认配置 (1, 3, 6, 224, 224) 的延迟为 5.75 秒，CIDEr 分数为 82.5，可作为比较基线。

(i) **将跟踪输出长度**调整为 12，可将延迟降至 4.55 秒，但 CIDEr 分数也降至 74.19，因为较短的输出往往无法涵盖真实描述中的全部细节。(ii) **将输入帧率（FPS）**从 6 降至 2，可将延迟降至 4.78 秒，但由于时间覆盖范围不足，性能会大幅下降（CIDEr：50.89）。(iii)**将帧分辨率**降至 (112, 112)，可将延迟降至 3.29 秒，性能仅轻微下降（CIDEr：75.64），体现了有效的权衡。(iv)**将束大小**降至 1，延迟为 3.07 秒，但由于生成过程中的搜索多样性有限，性能会严重下降（CIDEr：40.98）。(v) **将批大小**增至 4，会使延迟增至 22.35 秒，

#	由原始模型生成的字幕 全精度（float32）模型	由量化模型生成的字幕 量化（int8）模型
1	一个人正在包装礼物	一个人正在包装礼物
2	向相机展示城市景观	正在展示一座城市
3	一个人正在制作玉米饼	一个人正在制作玉米饼
4	一位女性正在拆开一个盒子	一位女性打开一个盒子
5	一位女性正在打开一个盒子	一个人正在打开一个盒子
6	一位女性正在祈祷	一位女性正在把蜡烛放在一尊雕像上
7	一个女孩正在读一本书	一个女孩正在读一本书
8	一名女性正在阅读一本书	一名女性正在阅读
9	向相机展示一张街头集市的照片	嘉年华的影像
10	拍摄城市时将相机向上转动	拍摄城市时将相机向上转动
11	一群人在户外演奏乐器	一群人在唱歌并演奏乐器
12	一个人在做饭	一个人在做饭

表 4：原始全精度 float32 mPLUG模型与量化 int8 视频语言模型（修改后的 mPLUG2+）生成的字幕（视频的文本描述）对比。可以看到细微但含义相同的差异，这表明量化后字幕质量得以保留。

输入形状 （批大小、通道数、FPS、宽度、高度）	延迟	CIDEr 分数
（1, 3, 6, 224, 224）：默认情况	5.57	82.5
(1, 3, 2, 224, 224)	4.78	50.89
(1, 3, 6, 112, 112)	3.29	75.64
(4, 3, 6, 224, 224)	22.35	82.5
(1, 3, 6, 224, 224): 生成器束宽 = 1	3.07	40.98
(1, 3, 6, 224, 224): 跟踪输出长度 = 12	4.55	74.19

表5：改变输入尺寸和超参数，以衡量其对延迟和性能的影响。所有实验均在 Google Pixel 5a 上使用 MSR-VTT 测试数据集进行视频组装任务测量。默认输入形状为 (1, 3, 6, 224, 224)，跟踪输出长度为 23，生成时的束大小为 4。

同时保持与默认设置相同的 CIDEr 分数（82.5），这表明批处理在受限硬件上并无速度优势。这些结果凸显了调整输入和解码设置以平衡延迟与质量的重要性，尤其是在移动端部署场景中。

F.2 存储大小

表6显示，视频语言模型 mPLUG+ 和句子转换器 all-MiniLM-L6-v2 的总存储大小为 1.28GB，其中视频语言模型占 1.23GB，句子转换器占 53MB。这种紧凑的大小确保模型能够存储在大多数普通智能手机上，从而支持在设备端部署用于视频字幕生成和检索的端到端流水线。

	存储大小 （兆字节）
视频语言模型（mPLUG+）	1230
句子转换器（all-MiniLM-L6-v2）	53
总计	1283

表6：用于视频组装、视频字幕生成和视频检索端到端任务流水线的视频语言模型和句子转换器的**存储大小（兆字节）**。这些流水线所需的总存储大小为 1.28GB，大多数普通智能手机都能够满足这一要求。

F.3 量化 SentenceTransformer

表 7 显示，all-MiniLM-L6-v2 SentenceTransformer 的全精度版本在语义文本相似度基准（STS-B）数据集的测试划分上，皮尔逊相关系数为 0.83。量化版本的表现略逊，皮尔逊相关系数为 0.81；不过，定性分析（见附录 E）表明，量化导致的性能下降对端到端性能的影响可以忽略不计。

F.4 内存占用

本节旨在量化 ATOM 的内存占用，并说明其以复用为中心的设计不会超出移动设备紧张的随机存取存储器预算。表 8 报告了视频语言流水线在采用和不采用模块化复用时的峰值内存使用量。两种配置仅相差 40MB：基线为 5.28GB，ATOM 为 5.32GB；增加幅度仅为 0.76%，微不足道，仍可轻松满足主流智能手机的 6 GB 随机存取存储器容量。额外的 40MB 来自 ATOM 下略高的激活值（0.479GB，而顺序基线为 0.297GB），这是因为编码器和解码器可以同时处于活动状态。这一开销被其他分配量的大幅下降所抵消；这些分配量是模型加载期间创建的临时缓冲区。最大内存消耗

SentenceTransformer 变体	皮尔逊相关系数（↑）
全精度	0.8287
量化	0.8156

表 7：句子转换器 all-MiniLM-L6-v2 在语义文本相似度基准（STS-B）数据集上的全精度版本与量化版本性能比较。我们的移动流水线使用该句子转换器，查找与用户查询最相关的视频的文本表示。可以看到，量化版本的性能略有下降。虽然 ATOM 能在移动流水线中容纳全精度模型，但我们观察到，使用量化句子转换器时，端到端性能没有下降。

	最大内存消耗（以 GB 为单位）	
	未采用模块化和复用	ATOM
模型权重	4.623	4.623
输入	0.005	0.005
激活值（所有模型或模块中的最高值）	0.297	0.479
其他（临时变量在模型加载中）	0.352	0.210
峰值内存消耗	5.277	5.317
单次流水线运行的		

表 8：**内存占用（单位：GB，越低越好）**用于**videoassembly**任务流水线的视频语言模型的内存占用。采用 ATOM 的流水线为 ATOM 消耗 5.3 GB 的随机存取存储器，这对于配备 6 GB 随机存取存储器的近期普通智能手机而言是可行的。与未采用模块化和复用的实现相比，ATOM 的内存使用量仅增加 0.76%，这是因为模型加载期间产生的临时分配减少了 40.34%。

模块名称	参数数量	参数大小（兆字节）	占总量的百分比 Size
线性层	1236M	3689	80.82%
嵌入层	161M	487	10.67%
三维卷积	51M	194	4.25%
非动态地可量化线性层	49M	193	4.22%
层归一化	0.4M	2	0.04%

表 9：PLUG2+ 峰值内存消耗的逐层细分。

与临时分配相关的内存占用减少了 40.34%（ATOM 为 0.210GB，而顺序基线为 0.352GB），这是因为重型模块始终驻留在内存中。两个变体的模型权重和输入内存保持不变。

延迟–内存权衡。 尽管 ATOM 为实现并行性牺牲了额外的 40MB 随机存取存储器，但端到端延迟降低了 27–33%（见第 5.3 节），因此这种权衡非常有利。若要进一步节省内存，就必须将量化精度降至 8 位以下，但我们发现这样会导致准确性无法接受地下降。相较之下，ATOM 优先采用复用来最大限度地提升速度，同时将内存占用控制在大众消费级手机实际可用的 6GB 范围内。

F.5 逐层内存细分

本节对 ATOM 中使用的视频语言模型 mPLUG2+ 各层的内存占用进行了更细致的分析。通过这一细分，我们得以识别内存总占用的主要来源。了解内存细分对于确定进一步优化的方向至关重要。

表 9 显示，线性层和嵌入层占据了内存使用量的大部分，分别占总大小的 80.82%（3,689MB）和 10.67%（487MB）。这些层在文本解码器中得到广泛使用，因此蕴含着显著的优化机会。

通过复用文本解码器，ATOM 有效降低了内存占用和执行延迟，展现了这一架构选择的高效性。这一发现进一步凸显了在资源受限环境中，针对深度学习流水线进行定向优化的重要性。

F.6 扩展结果：DiDeMo 数据集上的视频检索

为补充第 5.2 节中关于视频检索性能的主表（表 2），表 10 展示了完整的 DiDeMo 检索结果，凸显了 mPLUG2+ 即使在量化后仍能保持出色的性能。

F.7 Pixel 5a 上的能耗估算

为量化 ATOM 在普通移动设备上的能效优势，我们在 Google Pixel 5a 上计算了视频语言处理过程中的能耗降低幅度。

模型变体	DiDeMo		
	R@1	R@5	R@10
未量化（32位浮点权重）			
mPLUG2	55.1	78.3	84.6
mPLUG2+	60.0	86.2	90.1
使用量化（8位整数权重）			
mPLUG2	53.7	75.1	80.8
mPLUG2+	58.7	83.1	88.4

表 10：原始 mPLUG2（使用 BERT 文本解码器）与我们修改后的 mPLUG2+ （使用 Llama 文本解码器）在 DiDeMo 数据集上执行视频检索任务时，量化版本和全精度版本的**性能** 比较。所有 3 项指标 R@1（Recall@1）、R@5（Recall@5）和 R@10（Recall@10）均为“越高越好”。与文本解码器较弱的原始 mPLUG 相比，ATOM 的 mPLUG2+ 即使经过量化，性能下降幅度也较小。

• **CPU 功耗：** Pixel 5a 的骁龙 765G CPU 在视频语言模型推理典型的中等计算负载下，功耗约为 2 瓦。

• **DRAM 功耗：** LPDDR4X RAM 在活跃工作负载下以 6GB 利用率运行时，内存功耗约为 0.2 瓦。

• **延迟：** 原始流水线执行延迟为 $T_{\text{原始}}$ =10 秒（示例值）。ATOM 可将延迟降低 27–33%，使新的延迟达到 $T_{\text{新}} \approx 6.7$–7。3 秒。

• **内存占用：** ATOM 的峰值 RAM 使用量基本保持不变：5.277GB（基线）与 5.317GB（+0.76%）。增幅仅为 0.76%，因此我们假设 DRAM 功耗仍为 0.2W。

计算：

$$\begin{aligned}\text{Original Energy} &= (P_{\text{CPU}} + P_{\text{DRAM}}) \times T_{\text{orig}} \\ &= (2.0\,\text{W} + 0.2\,\text{W}) \times 10\,\text{s} = 22\,\text{J}\end{aligned}$$

$$\text{调整后的 CPU 功耗 } = 2.0\text{W } \times (1 - 0.27) \approx 1.46\,\text{W}$$

（假设 CPU 功耗随延迟成比例变化）

$$\text{调整后的 DRAM 功耗 } = 0.2\text{W } \times (1 - 0.057) \approx 0.19\,\text{W}$$

$$\begin{aligned}\text{New Energy} &= (1.46\,\text{W} + 0.19\,\text{W}) \times 7.0\,\text{s} \\ &= 11.9\,\text{J}\end{aligned}$$

$$\therefore \text{Energy Savings} = \frac{22 - 11.9}{22} \times 100\% \approx 46\%$$

此估算假设 CPU 功耗随延迟降低而线性缩放，内存功耗随内存占用减少而线性缩放，这对于短时、计算密集型工作负载而言是合理的。

影响： 每次流水线执行的能耗几乎减半，可显著延长电池续航时间

在重复执行视频语言任务时的续航时间。大部分节能来自 ATOM 的复用中心设计，该设计缩短了 CPU 活跃时间；内存功耗几乎没有变化，因为峰值占用仅增加了 0.76%。

G 超参数选择。

为了在字幕生成任务上训练 mPLUG2+模型，我们对学习率 {1e- 3、1e-4、1e-5、**2e-5**、5e-5、1e-6} 进行了网格搜索，发现 2e-5 在稳定性和性能之间取得了最佳平衡。训练超过 10 轮次会导致过拟合，因此我们使用了第 **10** 轮次的检查点。我们还评估了批次大小 {32, 64, **128**, 256}；批大小为 128 时，在训练时间和 CIDEr 分数之间取得了最佳平衡。其他视频相关超参数的影响在消融研究（附录 F.1 ）中进行了分析。

参考文献

Bain, M.; Nagrani, A.; Varol, G.; and Zisserman, A. 2021. Frozen in Time：一种用于端到端检索的联合视频和图像编码器。载于*IEEE*国际计算机视觉会议。Chen, D. L.; and Dolan, W. B. 2011. 收集用于释义评估的高度并行数据。载于计算语言学协会第*49*届年会论文集 *(ACL-2011)*。Chen, T.-S.; Siarohin, A.; Menapace, W.; Deyneka, E.; Chao, H.-w.; Jeon, B. E.; Fang, Y.; Lee, H.-Y.; Ren, J.; Yang, M.-H.; et al. 2024年。Panda-70m：使用多个跨模态教师为 7000 万个视频生成字幕。载于*IEEE/CVF*计算机视觉与模式识别会议论文集。Chu, X.; Qiao, L.; Lin, X.; Xu, S.; Yang, Y.; Hu, Y.; Wei, F.; Zhang, X.; Zhang, B.; Wei, X.; et al. 2023年。Mobilevlm：面向移动设备的快速、可复现且性能强大的视觉语言助手。*arXiv* 预印本 *arXiv:2312.16886*。Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; Yang, A.; Fan, A.; Goyal, A.; Hartshorn, A.; Yang, A.; Mitra, A.; Sravankumar, A.; Korenev, A.; Hinsvark, A.; Rao, A.; Zhang, A.; Rodriguez, A.; Gregerson, A.; Spataru, A.; Roziere, B.; Biron, B.; Tang, B.; Chern, B.; Caucheteux, C.; Nayak, C.; Bi, C.; Marra, C.; McConnell, C.; Keller, C.; Touret, C.; Wu, C.; Wong, C.; Ferrer, C. C.; Nikolaidis, C.; Allonsius, D.; Song, D.; Pintz, D.; Livshits, D.; Wyatt, D.; Esiobu, D.; Choudhary, D.; Mahajan, D.; Garcia-Olano, D.; Perino, D.; Hupkes, D.; Lakomkin, E.; AlBadawy, E.; Lobanova, E.; Dinan, E.; Smith, E. M.; Radenovic, F.; Guzm´an, F.; Zhang, F.; Synnaeve, G.; Lee, G.; Anderson, G. L.; Thattai, G.; Nail, G.; Mialon, G.; Pang, G.; Cucurell, G.; Nguyen, H.; Korevaar, H.; Xu, H.; Tou- vron, H.; Zarov, I.; Ibarra, I. A.; Kloumann, I.; Misra, I.; Evtimov, I.; Zhang, J.; Copet, J.; Lee, J.; Geffert, J.; Vranes, J.; Park, J.; Mahadeokar, J.; Shah, J.; van der Linde, J.; Bil- lock, J.; Hong, J.; Lee, J.; Fu, J.; Chi, J.; Huang, J.; Liu, J.; Wang, J.; Yu, J.; Bitton, J.; Spisak, J.; Park, J.; Rocca, J.; Johnstun, J.; Saxe, J.; Jia, J.; Al wala, K. V.; Prasad, K.; Upasani, K.; Plawiak, K.; Li, K.; Heafield, K.; Stone, K.; El- Arini, K.; Iyer, K.; Malik, K.; Chiu, K.; Bhalla, K.; Lakho- tia, K.; Rantala-Yeary, L.; van der Maaten, L.; Chen, L.; Tan, L.; Jenkins, L.; Martin, L.; Madaan, L.; Malo, L.; Blecher, L.; Landzaat, L.; de Oliveira, L.; Muzzi, M.; Pasupuleti, M.; Singh, M.; Paluri, M.; Kardas, M.; Tsi mpoukelli, M.; Oldham, M.; Rita, M.; Pavlova, M.; Kambadur, M.; Lewis, M.; Si, M.; Singh, M. K.; Hassan, M.; Goyal, N.; Torabi, N.; Bashlykov, N.; Bogoychev, N.; Chatterji, N.; Zhang, N.; Duchenne, O.; Cˌelebi, O.; Alrassy, P.; Zhang, P.; Li, P.; Vasic, P.; Weng, P.; Bhargava, P.; Dubal, P.; Krishnan, P.; Koura, P. S.; Xu, P.; He, Q.; Dong, Q.; Srinivasan, R.; Ganapathy, R.; Calderer, R.; Cabral, R. S.; Stojnic, R.; Raileanu, R.; Maheswari, R.; Girdhar, R.; Patel, R.; Sauvestre, R.; Polidoro, R.; Sumbaly, R.; Taylor, R.; Silva, R.; Hou, R.; Wang, R.; Hosseini, S.; Chennabasappa, S.; Singh, S.; Bell, S.; Kim, S. S.; Edunov, S.; Nie, S.; Narang, S.; Raparthy, S.; Shen, S.; Wan, S.; Bhosale, S.; Zhang, S.; Vandenhende, S.; Batra, S.; Whitman, S.; Sootla, S.; Collot, S.; Gururan- gan, S.; Borodinsky, S.; Herman, T.; Fowler, T.; Sheasha, T.; Georgiou, T.; Scialom, T.; Speckbacher, T.; Mihaylov, T.; Xiao, T.; Karn, U.; Goswami, V.; Gupta, V.; Ramanathan, V.; Kerkez, V.; Gonguet, V.; Do, V.; Vogeti, V.; Albiero, V.; Petrovic, V.; Chu, W.; Xiong, W.; Fu, W.; Meers, W.; Mar- tinet, X.; Wang, X.; Wang, X.; Tan, X. E.; Xia, X.; Xie, X.; Jia, X.; Wang, X.; Goldschlag, Y.; Gaur, Y.; Babaei, Y.; Wen, Y.; Song, Y.; Zhang, Y.; 李, Y.; Mao, Y.; Coudert, Z. D.; Yan, Z.; Chen, Z.; Papakipos, Z.; Singh, A.; Srivas- tava, A.; Jain, A.; Kelsey, A.; Shajnfeld, A.; Gangidi, A.; Victoria, A.; Goldstand, A.; Menon, A.; Sharma, A.; Boe- senberg, A.; Baevski, A.; Feinstein, A.; Kallet, A.; Sangani, A.; Teo, A.; Yunus, A.; Lupu, A.; Alvarado, A.; Caples, A.; Gu, A.; Ho, A.; Poulton, A.; Ryan, A.; Ramchandani, A.; Dong, A.; Franco, A.; Goyal, A.; Saraf, A.; Chowdhury, A.; Gabriel, A.; Bharambe, A.; Eisenman, A.; Yazdan, A.; James, B.; Maurer, B.; Leonhardi, B.; Huang, B.; Loyd, B.; Paola, B. D.; Paranjape, B.; Liu, B.; Wu, B.; Ni, B.; Han- cock, B.; Wasti, B.; Spence, B.; Stojkovic, B.; Gamido, B.; Montalvo, B.; Parker, C.; Burton, C.; Mejia, C.; Liu, C.; Wang, C.; Kim, C.; Zhou, C.; Hu, C.; Chu, C.-H.; Cai, C.; Tindal, C.; Feichtenhofer, C.; Gao, C.; Civin, D.; Beaty, D.; Kreymer, D.; 李, D.; Adkins, D.; Xu, D.; Testuggine, D.; David, D.; Parikh, D.; Liskovich, D.; Foss, D.; Wang, D.; Le, D.; Holland, D.; Dowling, E.; Jamil, E.; Montgomery, E.; Presani, E.; Hahn, E.; Wood, E.; Le, E.-T.; Brinkman, E.; Arcaute, E.; Dunbar, E.; Smothers, E.; Sun, F.; Kreuk, F.; Tian, F.; Kokkinos, F.; Ozgenel, F.; Caggioni, F.; Kanayet, F.; Seide, F.; Florez, G. M.; Schwarz, G.; Badeer, G.; Swee, G.; Halpern, G.; Herman, G.; Sizov, G.; Guangyi; Zhang; Lakshminarayanan, G.; Inan, H.; Shojanazeri, H.; Zou, H.; Wang, H.; Zha, H.; Habeeb, H.; Rudolph, H.; Suk, H.; Aspegren, H.; Goldman, H.; Zhan, H.; Damlaj, I.; Molybog, I.; Tufanov, I.; Leontiadis, I.; Veliche, I.-E.; Gat, I.; Weiss- man, J.; Geboski, J.; Kohli, J.; Lam, J.; Asher, J.; Gaya, J.- B.; Marcus, J.; Tang, J.; Chan, J.; Zhen, J.; Reizenstein, J.; Teboul, J.; Zhong, J.; Jin, J.; Yang, J.; Cummings, J.; Carvill, J.; Shepard, J.; McPhie, J.; Torres, J.; Ginsburg, J.; Wang, J.; Wu, K.; U, K. H.; Saxena, K.; Khandelwal, K.; Zand, K.; Matosich, K.; Veeraraghavan, K.; Michelena, K.; 李, K.; Ja- gadeesh, K.; Huang, K.; Chawla, K.; Huang, K.; Chen, L.; Garg, L.; A, L.; Silva, L.; Bell, L.; Zhang, L.; Guo, L.; Yu, L.; Moshkovich, L.; Wehrstedt, L.; Khabsa, M.; Avalani, M.; Bhatt, M.; Mankus, M.; Hasson, M.; Lennie, M.; Reso, M.; Groshev, M.; Naumov, M.; Lathi, M.; Keneally, M.; Liu, M.; Seltzer, M. L.; Valko, M.; Restrepo, M.; Patel, M.; Vyatskov, M.; Samvelyan, M.; Clark, M.; Macey, M.; Wang, M.; Hermoso, M. J.; Metanat, M.; Rastegari, M.; Bansal, M.; Santhanam, N.; Parks, N.; White, N.; Bawa, N.; Singhal, N.; Egebo, N.; Usunier, N.; Mehta, N.; Laptev, N. P.; Dong, N.; Cheng, N.; Chernoguz, O.; Hart, O.; Salpekar, O.; Kalinli, O.; Kent, P.; Parekh, P.; Saab, P.; Balaji, P.; Rittner, P.; Bon- trager, P.; Roux, P.; Dollar, P.; Zvyagina, P.; Ratanchandani, P.; Yuvraj, P.; Liang, Q.; Alao, R.; Rodriguez, R.; Ayub, R.; Murthy, R.; Nayani, R.; Mitra, R.; Parthasarathy, R.; 李, R.; Hogan, R.; Battey, R.; Wang, R.; Howes, R.; Rinott, R.; Mehta, S.; Siby, S.; Bondu, S. J.; Datta, S.; Chugh, S.; Hunt, S.; Dhillon, S.; Sidorov, S.; Pan, S.; Mahajan, S.; Verma,

S.; Yamamoto, S.; Ramaswamy, S.; Lindsay, S.; Lindsay, S.; Feng, S.; Lin, S.; Zha, S. C.; Patil, S.; Shankar, S.; Zhang, S.; Zhang, S.; Wang, S.; Agarwal, S.; Sajuyigbe, S.; Chin- tala, S.; Max, S.; Chen, S.; Kehoe, S.; Satterfield, S.; Govin- daprasad, S.; Gupta, S.; Deng, S.; Cho, S.; Virk, S.; Subra- manian, S.; Choudhury, S.; Goldman, S.; Remez, T.; Glaser, T.; Best, T.; Koehler, T.; Robinson, T.; Li, T.; Zhang, T.; Matthews, T.; Chou, T.; Shaked, T.; Vontimitta, V.; Ajayi, V.; Montanez, V.; Mohan, V.; Kumar, V. S.; Mangla, V.; Ionescu, V.; Poenaru, V.; Mihailescu, V. T.; Ivanov, V.; Li, W.; Wang, W.; Jiang, W.; Bouaziz, W.; Constable, W.; Tang, X.; Wu, X.; Wang, X.;Wu, X.; Gao,X.; Kleinman, Y.; Chen, Y.; Hu, Y.; Jia, Y.; Qi, Y.; Li, Y.; Zhang, Y.; Zhang, Y.; Adi, Y.; Nam, Y.; Yu; Wang; Zhao, Y.; Hao, Y.; Qian, Y.; Li, Y.; He, Y.; Rait, Z.; DeVito, Z.; Rosnbrick, Z.; Wen, Z.; Yang, Z.; Zhao, Z.; and Ma, Z. 2024年. Llama 3 模型家族。arXiv:2407.21783。

Hendricks, L. A.; Wang, O.; Shechtman, E.; Sivic, J.; Dar- rell, T.; and Russell, B. 2018年. 在自然语言处理经验方法 *(EMNLP)*中发表《使用时序语言定位视频中的片段》。

Jeong, S.; Kim, K.; Baek, J.; and Hwang, S. J. 2025年. VideoRAG：面向视频语料库的检索增强生成。arXiv:2501.05874。

Jo, W.; Lim, G.; Lee, G.; Kim, H.; Ko, B.; and Choi, Y.2024年. VVS：通过抑制无关帧实现视频到视频检索。*AAAI*人工智能会议论文集，38(3)。

Li, G.; and Li, Y. 2023年. SPOTLIGHT：聚焦视觉语言模型的移动 UI 理解。在第十一届国际学习表征会议上发表。

Long, X.; Ma, Z.; Hua, E.; Zhang, K.; Qi, B.; and Zhou, B. 2025年. 通过内置自回归搜索引擎实现检索增强视觉问答。在*AAAI*人工智能会议论文集，第39卷。

Luo, H.; Ji, L.; Zhong, M.; Chen, Y.; Lei, W.; Duan, N.; and Li, T. 2021. Clip4clip：端到端视频片段检索中片段的实证研究。*arXiv*预印本 *arXiv:2104.08860*。

Reimers, N.; and Gurevych, I. 2019年. Sentence-BERT：使用孪生BERT网络的句子嵌入表示。在*2019*年自然语言处理实证方法会议论文集。计算语言学协会。

Tahboub, K.; Gadgil, N. J.; and Delp, E. J. 2015年. 移动设备上的基于内容的视频检索：需要多少内容才足够？在*2015*年 *IEEE* 国际图像处理会议（*ICIP*）。

Tevissen, Y.; Guetari, K.; 和 Petitpont, F. 2024年. 面向大型视频库的检索增强生成. 收录于*2024*年 第*16*届国际人机系统交互会议 *(HSI)*。

torchao. 2024年. TorchAO：PyTorch 原生的训练到服务模型优化。

Wang, Y.; Li, K.; Li, X.; Yu, J.; He, Y.; Chen, G.; Pei, B.; Zheng,R.; Wang,Z.; Shi, Y.; 等. 2024年. Internvideo2：面向多模态视频理解的可扩展基础模型。在 ing 基础模型，用于多模态视频理解。载于 欧洲计算机视觉会议。施普林格。Wu, S.; Fei, H.; Qu, L.; Ji, W.; and Chua, T.-S. 2024年。Next- gpt：任意到任意多模态大语言模型。载于 第四十一届国际机器学习会议。Xu, D.; Yin, W.; Jin, X.; Zhang, Y.; Wei, S.; Xu, M.; and Liu, X. 2023年。LLMCad：快速且可扩展的设备端大语言模型推理。Xu, D.; Zhang, H.; Yang, L.; Liu, R.; Huang, G.; Xu, M.; and Liu, X. 2025年。借助NPU实现快速的设备端大语言模型推理。第*28*届年度移动计算与网络国际会议论文集。Xu, H.; Ye, Q.; Yan, M.; Shi, Y.; Ye, J.; Xu, Y.; Li, C.; Bi, B.; Qian, Q.; Wang, W.; Xu, G.; Zhang, J.; Huang, S.; Huang, F.; and Zhou, J. 2023年。mPLUG-2：跨文本、图像和视频的模块化多模态基础模型。载于第*40*届国际机器学习会议论文集。Xu, J.; Lan, C.; Xie, W.; Chen, X.; and Lu, Y. 2023年。基于检索的视频语言模型：高效的长视频问答。arXiv:2312.04931。Xu, J.; Mei, T.; Yao, T.; and Rui, Y. 2016年。MSR-VTT：用于连接视频与语言的大型视频描述数据集。IEEE计算机视觉与模式识别国际会议（CVPR）。Yan, D.; Li, P.; Li, Y.; Chen, H.; Chen, Q.; Luo, W.; Dong, W.; Yan, Q.; Zhang, H.; and Shen, C. 2025年。TG-LLaVA：通过可学习潜在嵌入实现的文本引导LLaVA。*AAAI*人工智能会议论文集，39。

Yang, G.; Lu, H.; Sun, Z.; 和 Lu, Z. 2023年. 使用文本脚本进行镜头检索与组装，以生成视频蒙太奇. 收录于*2023*年 *ACM*国际多媒体检索会议论文集，ICMR '23。计算机协会。

Yin, Z.; Ma, Y.; Cao, X.; Wang, B.; Chen, Q.; 和 Jiang, P. 2024年. 用于视频片段蒙太奇的文本-视频多粒度融合. arXiv:2412.09276. Zhang, H.; Jepson, A. D.; Mohomed, I.; Derpanis, K. G.; Zhang, R.; 和 Fazly, A. 2021. 移动设备上的个性化多模态视频检索. 收录于第*29*届 *ACM*国际多媒体会议论文集，MM '21。计算机协会。