# 메타 SAM 3 (Segment Anything Model 3) 모바일 최적화 및 경량화 기술 백서: 아키텍처 분석부터 ExecuTorch 배포까지의 포괄적 로드맵

## 서론: 프롬프트 기반 개념 분할(PCS)의 모바일 전환을 위한 기술적 도전과 기회

2025년 11월 메타(Meta)가 공개한 SAM 3(Segment Anything Model 3)는 컴퓨터 비전 분야, 특히 이미지 및 비디오 분할(Segmentation) 영역에서 기념비적인 진보를 이룩한 모델입니다.1 기존의 SAM 1과 SAM 2가 점, 박스, 마스크와 같은 기하학적 프롬프트에 반응하는 '시각적 분할(Promptable Visual Segmentation, PVS)'에 집중했다면, SAM 3는 텍스트 프롬프트를 통해 "줄무늬 고양이"나 "노란 스쿨버스"와 같은 의미론적 개념을 이해하고, 이미지와 비디오 전반에 걸쳐 해당 개념의 모든 인스턴스를 추적하는 '프롬프트 기반 개념 분할(Promptable Concept Segmentation, PCS)'을 도입했습니다.3 이러한 기능적 확장은 8억 4,800만 개(848M)에 달하는 파라미터와 정교한 통합 아키텍처를 통해 구현되었으며, 이는 NVIDIA H200과 같은 고성능 서버급 GPU 환경을 기준으로 설계되었습니다.2

본 보고서는 이러한 고성능 서버 기반의 파운데이션 모델인 SAM 3를, 제한된 연산 자원과 전력 허용량을 가진 모바일 디바이스(스마트폰, 엣지 AI 장치 등) 환경으로 이식하고 최적화하기 위한 포괄적인 기술 계획을 수립하는 것을 목적으로 합니다. 사용자의 요청에 따라 단순한 모델 변환 가이드를 넘어, 아키텍처 레벨의 경량화(Distillation), 수치적 정밀도 최적화(Quantization), 그리고 온디바이스 런타임(ExecuTorch) 배포에 이르는 전 과정을 심층적으로 분석하고, 실행 가능한 엔지니어링 로드맵을 제시합니다. 특히, 메타가 내부적으로 주력하고 있는 ExecuTorch 생태계와 최신 모바일 NPU(Neural Processing Unit) 가속 기술을 중심으로, 실시간 성능 확보를 위한 구체적인 전략을 논의할 것입니다.

## 제1장 SAM 3 아키텍처 심층 분석 및 모바일 병목 지점 진단

모바일 최적화 계획을 수립하기 위해서는 먼저 타겟 모델인 SAM 3의 내부 구조를 해부하고, 모바일 하드웨어에서 성능 저하를 유발할 수 있는 병목 지점(Bottleneck)을 명확히 식별해야 합니다. SAM 3는 기존 모델들과 달리 이미지 인식과 비디오 추적을 단일 백본으로 통합한 구조를 띠고 있으며, 이는 효율성 측면에서 장점인 동시에 모바일 이식에 있어 복잡성을 가중시키는 요인이기도 합니다.

### 1.1 통합 인식 인코더(Unified Perception Encoder)의 구조적 특성

SAM 3의 핵심은 이미지와 텍스트 입력을 처리하여 공통의 임베딩 공간으로 매핑하는 통합 인식 인코더(Perception Encoder, PE)에 있습니다.3 이 인코더는 MAE(Masked Autoencoder) 또는 Hiera 계열의 계층적 비전 트랜스포머(Vision Transformer, ViT) 아키텍처를 기반으로 하며, 시각적 정보와 의미론적 텍스트 정보를 정렬(Alignment)하는 역할을 수행합니다.4 서버 환경에서는 이러한 거대 트랜스포머가 높은 정확도를 보장하지만, 모바일 환경에서는 다음과 같은 치명적인 병목을 야기합니다.

첫째, **연산 복잡도()의 문제**입니다. ViT의 핵심인 셀프 어텐션(Self-Attention) 메커니즘은 입력 토큰 수(이미지 해상도)의 제곱에 비례하여 연산량과 메모리 사용량이 증가합니다. 고해상도 이미지를 처리해야 하는 분할 작업의 특성상, 모바일 CPU나 NPU에서 이러한 이차적 복잡도는 실시간 처리를 불가능하게 만드는 주요 원인입니다.

둘째, **메모리 대역폭(Memory Bandwidth)의 한계**입니다. 848M 파라미터를 가진 모델은 가중치(Weight) 로딩만으로도 약 3.4GB의 메모리를 점유합니다(FP32 기준).2 모바일 기기의 LPDDR5X 메모리 대역폭은 서버용 HBM(High Bandwidth Memory)에 비해 현저히 낮기 때문에, 매 프레임마다 거대한 가중치 행렬을 메모리에서 프로세서로 전송하는 과정에서 심각한 지연(Latency)이 발생합니다. 이는 단순한 연산 속도의 문제가 아니라, 데이터 이동에 소요되는 물리적 시간의 문제입니다.

### 1.2 DETR 기반 검출기(Detector)와 존재 헤드(Presence Head)

SAM 3는 객체 검출을 위해 DETR(DEtection TRansformer) 스타일의 아키텍처를 채택하고 있습니다.3 DETR은 학습 가능한 쿼리(Learnable Queries)를 사용하여 이미지 특징 맵에서 객체의 위치와 클래스를 예측합니다. 특히 SAM 3는 '존재 헤드(Presence Head)'라는 새로운 구성 요소를 도입하여, 특정 개념(예: "빨간 공")이 현재 장면에 존재하는지 여부를 먼저 판단하고, 존재할 경우에만 위치 추정(Localization)을 수행하는 방식을 취합니다.3 이 구조는 모바일 NPU 컴파일러 관점에서 난해한 점을 가지고 있습니다. 대부분의 모바일 NPU 컴파일러(CoreML, QNN 등)는 정적 형상(Static Shape)의 텐서 처리에 최적화되어 있습니다. 그러나 DETR 구조는 쿼리의 수나 검출된 객체의 수에 따라 후속 연산의 흐름이 달라질 수 있는 동적 특성을 가질 수 있어, 이를 정적 그래프로 변환하는 과정에서 상당한 최적화 노력이 필요합니다. 또한, 존재 헤드의 판단 로직이 조건부 분기(Conditional Branching)를 유발할 경우, 병렬 처리에 최적화된 NPU 파이프라인의 효율을 떨어뜨릴 수 있습니다.

### 1.3 메모리 기반 비디오 추적기(Memory-Based Video Tracker)

비디오 처리를 위해 SAM 3는 SAM 2로부터 계승된 메모리 뱅크(Memory Bank) 메커니즘을 사용합니다.3 이 시스템은 과거 프레임의 객체 특징을 저장해 두었다가 현재 프레임의 특징과 교차 어텐션(Cross-Attention)을 수행하여 객체를 추적합니다. 모바일 관점에서 이 메커니즘의 가장 큰 위험 요소는 **메모리 사용량의 선형적 증가**입니다. 비디오의 길이가 길어질수록 메모리 뱅크에 저장되는 데이터의 양이 누적되며, 이는 제한된 RAM을 가진 모바일 기기에서 앱 강제 종료(OOM, Out Of Memory)를 유발할 수 있습니다. 서버에서는 수십 기가바이트의 VRAM을 활용할 수 있지만, 스마트폰은 운영체제와 다른 앱들이 메모리를 공유하므로 단일 모델이 사용할 수 있는 메모리는 매우 제한적입니다. 따라서 모바일 최적화를 위해서는 단순히 모델을 경량화하는 것을 넘어, 비디오 메모리 관리 전략에 대한 근본적인 재설계가 필수적입니다.5

## 제2장 모바일 하드웨어 랜드스케이프 및 가속기 분석

성공적인 최적화 계획은 타겟 하드웨어에 대한 깊은 이해에서 출발합니다. 2025-2026년 시점의 최신 모바일 프로세서인 애플의 A18 Pro와 퀄컴의 스냅드래곤 8 Elite는 강력한 AI 가속기(NPU)를 내장하고 있으며, 이를 얼마나 효율적으로 활용하느냐가 성능의 핵심입니다.6

### 2.1 Apple A18 Pro와 Neural Engine (ANE)

애플의 A18 Pro 칩셋에 탑재된 Neural Engine은 초당 35조 회(35 TOPS)의 연산을 수행할 수 있으며, 특히 트랜스포머 모델 가속을 위한 하드웨어적 지원이 강화되었습니다.7

* **CoreML과의 통합:** ANE를 활용하기 위해서는 CoreML 포맷으로의 변환이 필수적입니다. CoreML은 최근 트랜스포머 아키텍처 지원을 강화하고 있으나, 여전히 일부 복잡한 연산(예: 5차원 이상의 텐서 조작, 특정 형태의 Gather/Scatter)은 CPU로 폴백(Fallback)되어 성능 저하를 일으킬 수 있습니다.
* **정밀도:** ANE는 FP16 및 Int8 연산에 최적화되어 있습니다. 특히 Int8 양자화(Quantization)를 적용할 경우, FP16 대비 2배 이상의 처리량과 전력 효율을 기대할 수 있습니다. 하지만 ONNX를 거쳐 CoreML로 변환하는 기존의 방식은 변환 과정에서의 불안정성과 연산자 호환성 문제가 보고되고 있어 8, PyTorch 생태계 내에서 직접 배포 가능한 경로(ExecuTorch)의 중요성이 커지고 있습니다.

### 2.2 Qualcomm Snapdragon 8 Elite와 Hexagon NPU

퀄컴의 최신 스냅드래곤 8 Elite는 45 TOPS 이상의 성능을 제공하는 Hexagon NPU를 탑재하고 있으며, 생성형 AI 모델 구동에 특화되어 있습니다.6

* **QNN (Qualcomm AI Engine Direct):** 퀄컴의 저수준 AI API인 QNN은 하드웨어 제어권이 강력하지만 개발 난이도가 높습니다. 최근 PyTorch ExecuTorch가 QNN 백엔드를 공식 지원하면서 10, PyTorch 모델을 직접 Hexagon NPU 바이너리로 컴파일하여 구동하는 워크플로우가 정립되었습니다.
* **메모리 아키텍처:** 스냅드래곤 NPU는 시스템 메모리(DDR)와 별도로 고속의 온칩 메모리(TCM, Tightly Coupled Memory)를 활용합니다. 모델의 가중치나 활성화 텐서(Activation Tensor)를 타일링(Tiling)하여 TCM 내에서 처리하도록 최적화하면, 외부 메모리 접근을 최소화하여 획기적인 속도 향상을 얻을 수 있습니다. 이는 양자화 전략과 밀접하게 연관됩니다.

### 2.3 모바일 GPU와 CPU의 역할

NPU가 AI 추론의 핵심이지만, GPU와 CPU의 역할도 무시할 수 없습니다.

* **모바일 GPU (Adreno, Apple GPU):** 병렬 연산에 강력하지만, NPU에 비해 전력 소모가 크고 다른 그래픽 작업(렌더링)과 자원을 공유해야 하는 단점이 있습니다.12 그러나 NPU가 지원하지 않는 특정 연산자(Custom Operators)를 처리하거나, 전처리/후처리 파이프라인을 가속화하는 데 유용합니다.
* **CPU (ARM Cortex):** 유연성이 가장 높지만 연산 성능은 낮습니다. 복잡한 제어 흐름(Control Flow)이나 매우 희소한(Sparse) 연산은 CPU에서 처리하는 것이 유리할 수 있습니다. ExecuTorch는 이러한 이종 프로세서 간의 작업 분배(Partitioning)를 지원하여, NPU-CPU 하이브리드 실행을 가능하게 합니다.13

## 제3장 전략적 최적화 1단계: 지식 증류(Knowledge Distillation) 기반 아키텍처 경량화

SAM 3의 848M 파라미터를 그대로 양자화하여 모바일에 올리는 것은 물리적으로 불가능에 가깝습니다. 따라서 첫 번째이자 가장 중요한 단계는 모델의 구조 자체를 모바일에 적합한 형태로 교체하는 **아키텍처 경량화**입니다. 이를 위해 **EfficientSAM3** 프로젝트에서 제안한 '점진적 계층적 증류(Progressive Hierarchical Distillation, PHD)' 방법론을 채택하여 적용할 것을 제안합니다.14

### 3.1 학생 모델(Student Model) 백본 선정 전략

거대 트랜스포머인 교사 모델(Teacher Model, SAM 3)의 지식을 흡수할 학생 모델(Student Model)은 모바일 친화적인 구조를 가져야 합니다. 단순히 레이어 수를 줄인 ViT가 아니라, 모바일 하드웨어의 특성(메모리 접근 패턴, 캐시 효율성)을 고려하여 설계된 아키텍처를 선정해야 합니다.

1. **RepViT (Latency 최우선 추천):** RepViT는 구조적 재매개변수화(Structural Re-parameterization) 기술을 사용하여, 학습 시에는 복잡한 다중 분기 구조를 가지지만 추론 시에는 단순한 합성곱(Convolution) 구조로 융합되는 아키텍처입니다.15 이는 ViT의 전역적 수용 영역(Global Receptive Field) 특성을 모사하면서도, 모바일 NPU/GPU에서 극도로 빠른 추론 속도를 제공합니다. 실시간 비디오 처리가 목표라면 RepViT가 최적의 선택입니다.
2. **TinyViT (정확도 중심 추천):** TinyViT는 경량화된 어텐션 메커니즘을 사용하는 소형 ViT입니다.14 CNN 기반의 RepViT보다 의미론적 정보(Text-Visual Alignment)를 보존하는 데 유리할 수 있어, 복잡한 텍스트 프롬프트에 대한 정확도가 중요하다면 TinyViT를 고려할 수 있습니다.
3. **MobileCLIP (텍스트 인코더 경량화):** SAM 3의 텍스트 인코더 또한 경량화 대상입니다. 이를 354M 파라미터에서 약 40-60M 수준의 MobileCLIP 변형 모델로 교체하여 증류해야 합니다.16 이는 전체 파이프라인의 메모리 사용량을 줄이는 데 크게 기여합니다.

### 3.2 3단계 점진적 증류 프로세스 (Three-Stage PHD)

EfficientSAM3의 방법론에 따라, 전체 모델을 한 번에 학습시키는 대신 세 단계로 나누어 증류를 수행합니다. 이는 각 구성 요소(인코더, 메모리, 전체 파이프라인)가 교사 모델의 복잡한 동작을 단계적으로 학습하도록 유도하여 성능 저하를 최소화합니다.14

#### 1단계: 이미지 레벨 인코더 증류 (Encoder Distillation)

* **목표:** SAM 3의 거대 PE가 생성하는 이미지 임베딩과 텍스트 임베딩의 정렬 관계를 학생 백본(RepViT/MobileCLIP)이 모방하도록 학습합니다.
* **데이터셋:** SA-1B 데이터셋의 일부(예: 1%~5%)를 사용합니다.16 SA-1B는 1,100만 장의 이미지와 11억 개의 마스크를 포함하는 방대한 데이터셋으로, 이를 통해 일반적인 객체 분할 능력을 전이시킵니다.
* **학습 전략:** '프롬프트-인-더-루프(Prompt-in-the-Loop)' 방식을 적용합니다. 단순히 특징 맵(Feature Map) 간의 MSE(Mean Squared Error) 손실만 사용하는 것이 아니라, 학생 모델이 생성한 임베딩으로 마스크를 예측하고, 이를 교사 모델의 예측 마스크와 비교하는 엔드-투-엔드 손실을 추가합니다. 이는 학생 모델이 특징 맵의 모든 픽셀을 완벽히 복제하기보다, 분할 작업에 중요한 핵심 특징에 집중하도록 돕습니다.

#### 2단계: 시간적 메모리 증류 및 압축 (Temporal Memory Distillation)

* **목표:** 비디오 추적 시 메모리 사용량을 고정하기 위해, SAM 3의 덴스(Dense) 메모리 뱅크를 **Perceiver Resampler** 기반의 압축 메모리 모듈로 대체하고 학습합니다.14
* **데이터셋:** SA-V (Segment Anything Video) 데이터셋을 활용합니다.
* **핵심 기술:** Perceiver Resampler는 가변 길이를 가진 과거 프레임들의 특징(개)을 고정된 개수(, 예: 64개)의 잠재 토큰(Latent Tokens)으로 압축합니다. 이를 통해 비디오 길이가 아무리 길어져도 추적기가 참조해야 할 메모리의 크기는 개로 유지되므로, 모바일 디바이스에서의 메모리 안정성을 보장할 수 있습니다. 학생 모델은 교사 모델(SAM 3)이 사용하는 풍부한 메모리 정보를 이 압축된 토큰들만으로 얼마나 잘 복원하고 추적에 활용할 수 있는지를 학습하게 됩니다.

#### 3단계: 엔드-투-엔드 개념 분할 미세 조정 (End-to-End PCS Fine-Tuning)

* **목표:** 경량화된 이미지 인코더, 텍스트 인코더, 압축 메모리 모듈을 결합하여, 텍스트 프롬프트 기반의 개념 분할(PCS) 성능을 최종적으로 최적화합니다.
* **데이터셋:** 메타가 공개한 SA-Co (Segment Anything with Concepts) 벤치마크 데이터셋을 사용합니다.3
* **학습 전략:** 이 단계에서는 전체 파이프라인을 동결 해제(Unfreeze)하고 미세 조정(Fine-Tuning)을 수행합니다. 특히 '존재 헤드(Presence Head)'와 디코더가 경량화된 백본의 특징 맵에 적응하도록 하여, "빨간 옷을 입은 사람"과 "파란 옷을 입은 사람"을 구분하는 미세한 의미론적 차이를 학습시킵니다.

## 제4장 전략적 최적화 2단계: 수치 정밀도 최적화 (Quantization & Precision Engineering)

아키텍처 경량화를 통해 모델의 크기를 줄였다면, 다음은 수치 정밀도를 낮추어 연산 속도를 가속화하고 모델 용량을 더욱 압축하는 단계입니다. 단순히 FP32를 Int8로 변환하는 것을 넘어, **ExecuTorch** 및 **torchao** 라이브러리를 활용한 정교한 양자화 전략이 필요합니다.

### 4.1 TorchAO (PyTorch Architecture Optimization) 활용

메타는 PyTorch 네이티브 양자화 라이브러리인 torchao를 통해 최신 양자화 기법들을 제공하고 있으며, 이는 ExecuTorch와 긴밀하게 통합되어 있습니다.18 SAM 3 최적화를 위해 다음과 같은 torchao 레시피를 적용할 것을 권장합니다.

1. **가중치 전용 양자화 (Weight-Only Quantization - Int4 Group-wise):**
   * **적용 대상:** RepViT 백본 및 트랜스포머 디코더의 선형(Linear) 레이어 가중치.
   * **기술 상세:** 가중치를 4비트 정수로 양자화하되, 채널 전체가 아닌 128개 또는 256개 단위의 그룹(Group)으로 나누어 스케일링 팩터(Scaling Factor)를 할당합니다. 이는 거대 언어 모델(LLM)에서 검증된 방식으로, 모델 크기를 FP16 대비 1/4로 줄이면서도 정확도 손실을 최소화합니다.20 모바일 디바이스의 제한된 저장 공간과 메모리 대역폭 문제를 해결하는 핵심 기술입니다.
2. **동적 활성화 양자화 (Dynamic Activation Quantization - Int8):**
   * **적용 대상:** 런타임에 값이 결정되는 활성화 텐서(Activation Tensor).
   * **기술 상세:** 입력 데이터에 따라 동적으로 스케일링 팩터를 계산하여 활성화 값을 8비트 정수로 변환합니다. 이는 트랜스포머의 어텐션 연산과 같이 값의 범위(Outlier)가 큰 레이어에서 정적 양자화(Static Quantization)보다 높은 정확도를 보장합니다.18

### 4.2 양자화 인지 학습 (Quantization-Aware Training, QAT)

PTQ(Post-Training Quantization)는 빠르지만, 4비트 수준의 과감한 양자화에서는 성능 저하가 발생할 수 있습니다. 특히 SAM 3와 같이 픽셀 단위의 정밀한 예측이 필요한 모델에서는 그 영향이 클 수 있습니다. 따라서 **QAT**를 적용하는 것이 필수적입니다.22

* **워크플로우:** 3단계 미세 조정(Fine-Tuning) 과정에서 torchao의 QAT API를 사용하여 가짜 양자화(Fake Quantization) 노드를 삽입합니다. 이를 통해 모델은 학습 과정에서 양자화로 인한 노이즈를 미리 경험하고, 이에 적응하는 방향으로 가중치를 업데이트하게 됩니다. 이는 최종 배포 모델의 정확도를 FP32 원본에 근접하게 유지하는 데 결정적인 역할을 합니다.

## 제5장 전략적 최적화 3단계: ExecuTorch 기반 배포 런타임 엔지니어링

최적화된 모델을 모바일 기기에서 구동하기 위한 런타임 환경으로 **ExecuTorch**를 선정합니다. 이는 메타가 PyTorch Mobile을 대체하기 위해 개발한 차세대 엣지 AI 런타임으로, SAM 3와 같은 복잡한 파이토치 모델을 수정 없이 효율적으로 배포할 수 있는 최적의 경로입니다.13

### 5.1 ExecuTorch 배포 파이프라인 (Lowering Pipeline)

ExecuTorch는 파이토치 모델을 기기에서 실행 가능한 바이너리 형태(.pte)로 변환하는 'Lowering' 과정을 거칩니다.24

1. **Export (내보내기):** torch.export.export() 함수를 사용하여 PyTorch 모델의 연산 그래프를 ATen 연산자 단위로 캡처합니다. 이 단계에서 동적 제어 흐름(if문, loop문)이 그래프 내에 포함됩니다.
2. **To Edge (엣지 변환):** to\_edge() 함수를 통해 그래프를 엣지 디바이스용 IR(Intermediate Representation)인 'Edge Dialect'로 변환합니다. 이 과정에서 불필요한 연산자가 제거되고 메모리 레이아웃이 최적화됩니다.
3. **Partition & Delegate (분할 및 위임):** 가장 중요한 단계입니다. 모델의 각 부분을 어떤 하드웨어 가속기(Backend)에서 실행할지 결정합니다.
   * **NPU 위임:** 합성곱(Conv), 행렬곱(MatMul) 등 연산 집약적인 부분은 CoreML(iOS) 또는 QNN(Android) 파티셔너를 통해 NPU로 위임합니다.
   * **CPU 폴백:** NPU가 지원하지 않는 복잡한 연산이나, DETR의 동적 쿼리 처리 부분 등은 XNNPACK 파티셔너를 통해 CPU에서 처리하도록 설정합니다. ExecuTorch는 이러한 이종 연산 장치 간의 협업을 매끄럽게 지원합니다.
4. **Memory Planning (메모리 계획):** 실행 시 필요한 모든 텐서의 생명 주기를 분석하여, 정적 메모리 블록을 할당합니다. 이는 런타임 중의 동적 메모리 할당(malloc) 오버헤드를 제거하여 추론 속도를 안정화합니다.

### 5.2 플랫폼별 백엔드 최적화 전략

* **iOS (CoreML Backend):** 애플 기기 배포를 위해 CoreMLPartitioner를 사용합니다. 이때, 트랜스포머의 Multi-Head Attention 연산이 ANE(Apple Neural Engine)에서 효율적으로 실행되도록 torch.nn.functional.scaled\_dot\_product\_attention (SDPA) 연산자를 CoreML의 해당 레이어로 매핑하는 것이 중요합니다.25
* **Android (QNN Backend):** 퀄컴 칩셋을 위해 QnnPartitioner를 사용합니다. 스냅드래곤 8 Elite의 경우, HTP(Hexagon Tensor Processor) 백엔드를 타겟으로 설정하고, 양자화된 모델을 전달하여 DSP 레벨의 가속을 활성화해야 합니다.10 QNN 백엔드는 초기 설정이 까다로우므로, 퀄컴 AI Hub에서 제공하는 튜토리얼과 Docker 이미지를 활용하여 환경을 구축하는 것이 권장됩니다.

## 제6장 단계별 구현 로드맵 및 엔지니어링 계획 (Implementation Plan)

위에서 논의한 기술적 전략을 바탕으로, 실제 프로젝트 수행을 위한 12주 단위의 상세 실행 계획을 수립합니다.

### 1단계: 환경 구축 및 베이스라인 수립 (1-2주차)

* **목표:** SAM 3 원본 모델의 추론 파이프라인 구축 및 데이터셋 준비.
* **세부 활동:**
  + 메타의 공식 GitHub 저장소 클론 및 SAM 3 모델(848M) 다운로드.3
  + PyTorch Nightly 빌드, ExecuTorch, TorchAO 라이브러리 설치 및 개발 환경 설정.13
  + SA-1B 데이터셋(1%) 및 SA-Co 데이터셋 다운로드 및 전처리.
  + 원본 SAM 3 모델의 추론 결과를 저장하여, 이후 증류 학습 시 '교사(Teacher)' 데이터로 활용할 파이프라인 구축.

### 2단계: 아키텍처 경량화 및 증류 학습 (3-6주차)

* **목표:** EfficientSAM3 방법론을 적용한 경량 학생 모델(Mobile-SAM 3) 개발.
* **세부 활동:**
  + **백본 교체:** RepViT-M2.3(또는 M1.1)으로 이미지 인코더 교체 및 ImageNet 사전 학습 가중치 로드.
  + **텍스트 인코더 교체:** MobileCLIP-S1 모델 통합.
  + **1단계 학습 (Encoder):** SA-1B 데이터를 사용하여 이미지 인코더 증류. 마스크 손실(Mask Loss)과 특징 정렬 손실(Feature Alignment Loss) 최소화.
  + **중간 평가:** SA-Co 검증 세트에서 교사 모델 대비 성능 85% 이상 달성 여부 확인.

### 3단계: 비디오 메모리 압축 및 전체 미세 조정 (7-8주차)

* **목표:** 모바일 비디오 처리를 위한 메모리 모듈 최적화.
* **세부 활동:**
  + **Perceiver Resampler 구현:** 입력 프레임 토큰을 64개의 잠재 토큰으로 압축하는 모듈 개발.
  + **2단계 학습 (Memory):** SA-V 데이터셋을 사용하여 메모리 모듈 학습.
  + **3단계 학습 (End-to-End):** 전체 파이프라인(백본+메모리+디코더)을 SA-Co 데이터로 미세 조정(QAT 적용 준비).

### 4단계: 양자화 및 정확도 복원 (9주차)

* **목표:** Int4/Int8 정밀도로 모델 변환 및 성능 검증.
* **세부 활동:**
  + torchao를 사용하여 가중치(Int4 Group-wise) 및 활성화(Int8 Dynamic) 양자화 적용.
  + 양자화된 모델의 정확도(mIoU) 측정.
  + 정확도 하락 폭이 2%를 초과할 경우, QAT(Quantization-Aware Training)를 1-2 에포크 수행하여 가중치 보정.22

### 5단계: ExecuTorch 변환 및 모바일 앱 통합 (10-12주차)

* **목표:** 온디바이스 실행 가능한 바이너리(.pte) 생성 및 데모 앱 빌드.
* **세부 활동:**
  + **Export & Partition:** iOS용 CoreML, Android용 QNN 파티셔너를 적용하여 .pte 파일 생성.
  + **런타임 테스트:** 데스크탑 환경에서 ExecuTorch 런타임으로 .pte 파일 로드 및 추론 테스트.
  + **앱 통합:** Android Studio(Kotlin/JNI) 및 Xcode(Swift/C++) 프로젝트에 ExecuTorch 라이브러리 및 모델 바이너리 탑재.
  + **성능 최적화:** 온디바이스 프로파일링 툴을 사용하여 메모리 누수 및 병목 구간 확인 및 수정.

## 결론 및 제언

SAM 3의 모바일 최적화는 단순한 포맷 변환이 아닌, 모델의 **재설계(Re-architecting)** 수준의 엔지니어링이 요구되는 도전적인 과제입니다. 본 보고서에서 제시한 **EfficientSAM3 기반의 점진적 증류**와 **ExecuTorch/TorchAO를 활용한 하드웨어 가속** 전략은 현재 기술 수준에서 가장 현실적이고 효과적인 접근법입니다.

특히, 8억 개가 넘는 파라미터를 가진 원본 모델을 모바일에서 구동하려는 시도는 메모리 대역폭의 물리적 한계로 인해 실패할 가능성이 매우 높습니다. 따라서 반드시 **학생 모델(Student Model)로의 증류**를 선행해야 하며, 이 과정에서 텍스트 프롬프트 처리 능력(PCS)을 보존하는 것이 핵심 성공 요인이 될 것입니다. 이 로드맵을 통해 귀하는 최신 파운데이션 모델의 강력한 기능을 손바닥 위의 디바이스에서 실시간으로 구현하는 혁신적인 성과를 거둘 수 있을 것입니다.

**참고 문헌 데이터베이스:**

* **Architecture & Methodology:** 1
* **Distillation Strategy:** 14
* **Quantization & TorchAO:** 18
* **ExecuTorch & Deployment:** 10
* **Hardware Insights:** 6
* **Benchmarks & Datasets:** 2

#### 참고 자료

1. 1038lab/sam3 - Hugging Face, 2월 8, 2026에 액세스, <https://huggingface.co/1038lab/sam3>
2. Segment Anything 3 Instance Segmentation Model: What is, How to ..., 2월 8, 2026에 액세스, <https://roboflow.com/model/segment-anything-3>
3. facebookresearch/sam3: The repository provides code for running inference and finetuning with the Meta Segment Anything Model 3 (SAM 3), links for downloading the trained model checkpoints, and example notebooks that show how to use the model. - GitHub, 2월 8, 2026에 액세스, <https://github.com/facebookresearch/sam3>
4. SAM 3: Segment Anything with Concepts - Ultralytics YOLO Docs, 2월 8, 2026에 액세스, <https://docs.ultralytics.com/models/sam-3/>
5. SAM 3: A Technical Deep Dive into Meta's Next-Generation Segmentation Model - Datature, 2월 8, 2026에 액세스, <https://datature.com/blog/sam-3-a-technical-deep-dive-into-metas-next-generation-segmentation-model>
6. NPU information for Apple and Snapdragon : r/LocalLLaMA - Reddit, 2월 8, 2026에 액세스, <https://www.reddit.com/r/LocalLLaMA/comments/1gy9wsx/npu_information_for_apple_and_snapdragon/>
7. Apple A18 Pro Processor - Benchmarks and Specs - NotebookCheck.net Tech, 2월 8, 2026에 액세스, <https://www.notebookcheck.net/Apple-A18-Pro-Processor-Benchmarks-and-Specs.891556.0.html>
8. ONNX Runtime and CoreML May Silently Convert Your Model to FP16 - Hacker News, 2월 8, 2026에 액세스, <https://news.ycombinator.com/item?id=46350075>
9. ONNX Runtime & CoreML May Silently Convert Your Model to FP16 (And How to Stop It), 2월 8, 2026에 액세스, <https://ym2132.github.io/ONNX_MLProgram_NN_exploration>
10. Run Llama 3 3B Instruct on Android (with Qualcomm AI Engine Direct Backend) — ExecuTorch 1.0 documentation, 2월 8, 2026에 액세스, <https://docs.pytorch.org/executorch/1.0/llm/build-run-llama3-qualcomm-ai-engine-direct-backend.html>
11. Building and Running ExecuTorch with Qualcomm AI Engine Direct Backend, 2월 8, 2026에 액세스, <https://docs.pytorch.org/executorch/0.3/build-run-qualcomm-ai-engine-direct-backend.html>
12. Unlocking Peak Performance on Qualcomm NPU with LiteRT - Google for Developers Blog, 2월 8, 2026에 액세스, <https://developers.googleblog.com/unlocking-peak-performance-on-qualcomm-npu-with-litert/>
13. pytorch/executorch: On-device AI across mobile, embedded and edge for PyTorch - GitHub, 2월 8, 2026에 액세스, <https://github.com/pytorch/executorch>
14. EfficientSAM3: Progressive Hierarchical Distillation for Video Concept Segmentation from SAM1, 2, and 3 - arXiv, 2월 8, 2026에 액세스, <https://arxiv.org/html/2511.15833>
15. RepViT: Revisiting Mobile CNN From ViT Perspective, 2월 8, 2026에 액세스, <https://ise.thss.tsinghua.edu.cn/mig/2024-9.pdf>
16. EfficientSAM3 compresses SAM3 into lightweight, edge-friendly models via progressive knowledge distillation for fast promptable concept segmentation and tracking. - GitHub, 2월 8, 2026에 액세스, <https://github.com/SimonZeng7108/efficientsam3>
17. EfficientSAM3: Progressive Hierarchical Knowledge Distillation from SAM1, SAM2 & SAM3, 2월 8, 2026에 액세스, <https://simonzeng7108.github.io/efficientsam3/>
18. Quantization Overview — ExecuTorch 1.0 documentation, 2월 8, 2026에 액세스, <https://docs.pytorch.org/executorch/1.0/quantization-overview.html>
19. pytorch/ao: PyTorch native quantization and sparsity for training and inference - GitHub, 2월 8, 2026에 액세스, <https://github.com/pytorch/ao>
20. torchao.quantization - PyTorch documentation, 2월 8, 2026에 액세스, <https://docs.pytorch.org/ao/stable/api_ref_quantization.html>
21. Quantization Overview — torchao 0.15 documentation, 2월 8, 2026에 액세스, <https://docs.pytorch.org/ao/stable/quantization_overview.html>
22. 1 Introduction - arXiv, 2월 8, 2026에 액세스, <https://arxiv.org/html/2507.16099v1>
23. Deploy YOLO26 on Mobile & Edge with ExecuTorch - Ultralytics YOLO Docs, 2월 8, 2026에 액세스, <https://docs.ultralytics.com/integrations/executorch/>
24. Exporting AI models on Android with XNNPACK and ExecuTorch - Software Mansion, 2월 8, 2026에 액세스, <https://blog.swmansion.com/exporting-ai-models-on-android-with-xnnpack-and-executorch-3e70cff51c59>
25. Accelerating On-Device ML Inference with ExecuTorch and Arm SME2 - PyTorch, 2월 8, 2026에 액세스, <https://pytorch.org/blog/accelerating-on-device-ml-inference-with-executorch-and-arm-sme2/>
26. quic/ai-hub-models - GitHub, 2월 8, 2026에 액세스, <https://github.com/quic/ai-hub-models>
27. EfficientSAM3: Progressive Hierarchical Distillation for Video Concept Segmentation from SAM1, 2, and 3 - arXiv, 2월 8, 2026에 액세스, <https://arxiv.org/html/2511.15833v1>
28. SAQ-SAM: Semantically-Aligned Quantization for Segment Anything Model - arXiv, 2월 8, 2026에 액세스, <https://arxiv.org/html/2503.06515v2>
29. EfficientSAM3: Quantum & Vision Efficiency - Emergent Mind, 2월 8, 2026에 액세스, <https://www.emergentmind.com/topics/efficientsam3>