# Gemma-Prune: A Multi-Stage Compression Pipeline for Deploying Gemma 3 4B Vision-Language Model on Mobile Devices

AtomGradient
https://github.com/AtomGradient/swift-gemma-cli

**Abstract**

We present GEMMA-PRUNE, a multi-stage model compression pipeline that reduces the Gemma 3 4B IT QAT vision-language model from 2.8 GB to 2.1 GB while preserving both text generation and image understanding capabilities. Our approach combines vocabulary pruning (262K→144K tokens), vision encoder quantization with dimension padding, text layer removal, image resolution reduction (896→672 pixels), dead neuron pruning, and weight splitting for lazy loading. Deployed on Apple Silicon via MLX Swift, the compressed model achieves 22% faster text generation (110 vs. 90 tokens/s), 3.4× faster image prompt processing (184 vs. 54 tokens/s), and 23% lower peak memory (2.2 GB vs. 2.9 GB for text-only inference). We identify critical failure modes in the compression process: vision layer removal destroys image understanding even with only 35 MB of savings, while reducing resolution to 448 px causes token repetition loops during generation. The optimal resolution of 672 px reduces vision attention compute by approximately 3× without quality degradation. Our pipeline and deployment tools are open-sourced to facilitate on-device VLM deployment research.

## 1 Introduction

The deployment of vision-language models (VLMs) on mobile devices presents a compelling but technically demanding frontier. Modern smartphones, particularly those equipped with Apple Silicon (iPhone 15 Pro, iPad Pro), offer neural engine capabilities and unified memory architectures that make on-device inference feasible in principle. However, the practical constraints are severe: the iPhone and iPad impose a hard limit of approximately 8 GB of unified RAM shared between the operating system, applications, and model weights, leaving only 4–5 GB available for model inference in typical usage scenarios.

Gemma 3 4B IT [Team et al., 2025] represents Google's state-of-the-art compact vision-language model, combining a SigLIP-based vision encoder [Zhai et al., 2023] with a 4-billion parameter text transformer. The model supports multimodal inputs—processing both text and images—making it suitable for a wide range of mobile applications including visual question answering, document understanding, and accessibility tools. The Quantization-Aware Training (QAT) variant at 4-bit precision [Nagel et al., 2022] reduces the model to 2.8 GB on disk, a significant reduction from the original bfloat16 representation, but runtime memory consumption of approximately 5.5 GB during image processing still exceeds practical mobile deployment budgets.

In this work, we present GEMMA-PRUNE, a systematic multi-stage compression pipeline that addresses this gap. Our approach is motivated by the observation that no single compression technique is sufficient; instead, we combine seven complementary optimization steps, each targeting a different source of redundancy:

1. **Vocabulary pruning**: The original vocabulary of 262,208 tokens contains extensive CJK, Arabic, and Cyrillic scripts unnecessary for English-only deployment. We prune to 144K tokens, saving 170 MB through compact embedding remapping.

2. **Vision fc2 quantization with dimension padding**: The SigLIP vision encoder's intermediate dimension (4304) is not divisible by any MLX-supported quantization group size. We introduce a zero-padding technique that enables 4-bit quantization while preserving mathematical equivalence.

3. **Text layer pruning**: We remove the three deepest transformer layers (31–33), exploiting the redundancy in late-stage feature refinement.

4. **Resolution reduction**: We reduce the vision encoder input from 896×896 to 672×672 pixels, cutting vision self-attention compute by ∼3×.

5. **Dead neuron pruning**: Activation profiling reveals that 60–100% of MLP neurons in layers 14–30 are effectively dead. We physically remove these neurons with careful alignment constraints.

6. **Weight splitting**: Separating language and vision weights enables lazy loading—text-only conversations load only 1.9 GB, with vision weights loaded on demand.

Critically, our work also documents negative results. We demonstrate that removing even four vision encoder layers (saving only 35 MB) completely destroys image understanding, and that reducing resolution below 672 px induces degenerate token repetition loops. These findings establish important boundaries for VLM compression.

The compressed model is deployed through `gemma-cli`, a Swift command-line tool built on Apple's MLX framework [Hannun et al., 2023], enabling streaming inference with real-time performance statistics. Our contributions are:

- A complete, reproducible 7-step compression pipeline reducing Gemma 3 4B from 2.8 GB to 2.1 GB with preserved multimodal capabilities.
- A dimension-padding technique for quantizing layers with prime-factor intermediate dimensions.
- Systematic failure mode analysis identifying critical thresholds in VLM compression.
- An open-source Swift deployment pipeline for Apple Silicon devices.

## 2 Related Work

### 2.1 Model Compression

Model compression encompasses a broad family of techniques for reducing the computational and memory footprint of neural networks. **Quantization** reduces the numerical precision of weights and activations, with post-training quantization (PTQ) [Frantar et al., 2023, Lin et al., 2024] and quantization-aware training (QAT) [Nagel et al., 2022] being the dominant paradigms. Our work starts from a QAT 4-bit model, applying further compression on top of an already-quantized baseline—a relatively underexplored regime.

**Structured pruning** removes entire neurons, attention heads, or layers from transformer architectures [Ma et al., 2023, Men et al., 2024]. Unlike unstructured pruning, structured approaches

yield immediate speedups without specialized sparse computation kernels. We employ both layer-level pruning (removing entire transformer blocks) and neuron-level pruning (removing MLP neurons based on activation statistics).

**Knowledge distillation** [Hinton et al., 2015] trains a smaller student model to mimic a larger teacher, and has been applied to language models [Sanh et al., 2019] and vision-language models [Fang et al., 2024]. We do not employ distillation in the current pipeline but identify it as a promising direction for quality recovery in future work.

### 2.2 Vision-Language Model Optimization

Vision-language models present unique compression challenges due to their heterogeneous architecture—typically a vision encoder, a projection layer, and a text decoder, each with different sensitivity to compression [Liu et al., 2024, Alayrac et al., 2022]. Recent work on LLaVA compression [Shang et al., 2024] has shown that vision encoders are significantly more sensitive to pruning than text decoders, a finding we corroborate in our experiments.

The SigLIP vision encoder [Zhai et al., 2023] used in Gemma 3 employs a standard Vision Transformer (ViT) [Dosovitskiy et al., 2021] architecture with $14 \times 14$ patches and learned position embeddings. Compressing the vision encoder requires careful handling of position embedding interpolation [Cai et al., 2023], which we address through 2D bilinear interpolation on the patch grid.

### 2.3 Mobile Deployment Frameworks

Apple's MLX framework [Hannun et al., 2023] provides a NumPy-compatible array library optimized for Apple Silicon's unified memory architecture. MLX Swift [Apple, 2024] extends this to native Swift applications, enabling direct integration with iOS and macOS apps. The unified memory model eliminates CPU-GPU data transfers, making it particularly suitable for large model inference. Alternative frameworks include Core ML [Apple, 2023] (which requires model conversion and has limited support for novel architectures) and llama.cpp [Gerganov, 2023] (which supports quantized inference but lacks native vision-language model support on Apple platforms).

### 2.4 Vocabulary Pruning

Multilingual language models allocate significant embedding capacity to scripts and languages unnecessary for specific deployment scenarios. Vocabulary pruning [Zheng et al., 2024] selectively removes unused tokens and compresses the embedding matrix. For models using SentencePiece [Kudo and Richardson, 2018] or BPE [Sennrich et al., 2016] tokenization, naive vocabulary pruning can break subword tokenization chains, leading to out-of-vocabulary tokens during inference. Our approach addresses this through comprehensive token collection that includes BPE-merged forms, inflected variants, and byte fallback tokens.

## 3 Method

We describe the GEMMA-PRUNE compression pipeline, which consists of seven sequential stages. Each stage produces a self-contained model checkpoint, enabling validation at every step. The pipeline is designed to be applied to the Gemma 3 4B IT QAT 4-bit model [Team et al., 2025], though the techniques are broadly applicable to similar vision-language architectures.

### 3.1 Architecture Overview

The Gemma 3 4B IT model comprises three components:

**Text Decoder.** A 34-layer transformer with grouped-query attention (GQA): 8 attention heads, 4 key-value heads, head dimension 256, hidden size 2560, and MLP intermediate size 10240. The activation function is `gelu_pytorch_tanh`. Each layer uses four layer normalization operations: input, post-attention, pre-feedforward, and post-feedforward. The model employs tied word embeddings (*i.e.*, the embedding matrix is shared between input and output).

**Vision Encoder.** A SigLIP [Zhai et al., 2023] Vision Transformer with 27 layers, hidden size 1152, intermediate size 4304, and 16 attention heads. Input images are processed at $896 \times 896$ resolution with $14 \times 14$ patches, yielding $64 \times 64 = 4096$ patch embeddings that are pooled to 256 image tokens.

**Multimodal Projector.** A linear projection from the vision encoder's hidden dimension (1152) to the text decoder's hidden dimension (2560), bridging the two modalities.

The QAT 4-bit variant quantizes all linear layers in the text decoder to 4-bit precision with group size 64, while the vision encoder remains in bfloat16 (with the exception of quantized position embeddings). The total model size is 2.8 GB stored as a single `safetensors` file.

### 3.2 Step 1: Vocabulary Pruning

The original Gemma 3 vocabulary contains 262,208 tokens, including extensive coverage of CJK ideographs, Arabic script, Cyrillic, Devanagari, and other writing systems. For English-only mobile deployment, the vast majority of these tokens are unused, yet each contributes a row to the embedding matrix $\mathbf{E} \in \mathbb{R}^{262208 \times d}$ (where $d = 2560$ is the hidden dimension in unquantized form, or its quantized equivalent).

#### 3.2.1 Token Collection Strategy

We construct a retained token set $\mathcal{T}$ through multiple collection passes:

1. **System dictionary**: All words from the macOS system dictionary (`/usr/share/dict/web2`), providing approximately 235,000 English word forms.
2. **Inflected forms**: For each dictionary word, we generate 26 morphological variants using common English suffixes (`-s`, `-ed`, `-ing`, `-er`, `-est`, `-ly`, `-tion`, `-ness`, `-ment`, `-able`, `-ible`, `-ful`, `-less`, `-ous`, `-ive`, `-al`, `-ize`, `-ise`, `-ity`, `-ence`, `-ance`, `-ure`, `-dom`, `-ship`, `-ward`, `-wise`).
3. **ASCII vocabulary scan**: All tokens in the original vocabulary that consist entirely of ASCII-printable characters (codes 32–126). This is critical for capturing BPE-merged subword tokens.
4. **Numeric tokens**: All tokens representing numbers, decimal points, and common numeric formats.
5. **Byte fallback tokens**: All 256 byte-level tokens used by SentencePiece for out-of-vocabulary character encoding.
6. **Special tokens**: Beginning-of-sequence, end-of-sequence, padding, vision tokens (`boi`=255999, `eoi`=256000, `image`=262144), and chat template tokens (`<start_of_turn>`, `<end_of_turn>`).

*[Placeholder: Bar chart showing token count by collection source. ASCII vocabulary scan contributes the most retained tokens beyond the base dictionary.]*

Figure 1: Distribution of retained tokens by collection source. The ASCII vocabulary scan (Pass 3) is essential for capturing BPE-merged subword tokens that do not appear in any dictionary.

#### 3.2.2 Critical Finding: BPE Token Coverage

An early version of our pipeline retained only ∼80K tokens based on dictionary words and their inflections. This approach produced severe generation quality degradation: the model would generate nonsensical outputs or fail to produce common English words. The root cause is that BPE tokenization creates merged tokens (*e.g.*, `_the`, `_and`, `_is`) that do not correspond to dictionary entries but are essential for fluent generation.

Adding the ASCII vocabulary scan (Pass 3) increased the retained set to ∼144K tokens and completely resolved the quality issue. This establishes a practical lower bound: for Gemma 3 with English deployment, the vocabulary cannot be reduced below approximately 144K tokens without degrading generation quality.

#### 3.2.3 Compact Embedding Construction

Given the retained token set $\mathcal{T}$ with $|\mathcal{T}| = N'$ tokens, we construct:

- A **compact embedding matrix** $\mathbf{E}' \in \mathbb{R}^{(N'+1)\times d}$, where row 0 is a zero vector (default for pruned tokens) and rows 1 through $N'$ contain the embeddings of retained tokens in their original order.
- A **token map** $\mathbf{M} \in \mathbb{Z}^V$, where $V = 262208$ is the original vocabulary size. For each token index $t \in [0, V)$:

$$\mathbf{M}[t] = \begin{cases} k & \text{if token } t \text{ is the } k\text{-th retained token} \\ 0 & \text{if token } t \text{ is pruned} \end{cases} \tag{1}$$

During inference, the embedding lookup becomes:

$$\mathbf{h} = \mathbf{E}'[\mathbf{M}[t]] \tag{2}$$

Since Gemma 3 uses tied word embeddings (`tie_word_embeddings: true`), the same token map is applied during output logit computation:

$$\text{logits}[t] = (\mathbf{E}'[\mathbf{M}[t]])^\top \mathbf{h}_{\text{final}} \tag{3}$$

The token map overhead is $262208 \times 4$ bytes $\approx 1\,\text{MB}$ (stored as `int32`), which is negligible compared to the embedding savings.

**Savings.** The original embedding (quantized at 4-bit with group size 64) occupies approximately 262208 rows. The compact embedding at 144257 rows reduces this by ∼45%, saving approximately 170 MB of disk and memory.

### 3.3 Step 2: Vision fc2 Quantization with Dimension Padding

The SigLIP vision encoder in Gemma 3 contains 27 transformer layers, each with an MLP block consisting of `fc1` (expansion) and `fc2` (projection) layers. While the text decoder is already 4-bit quantized, the vision encoder weights remain in bfloat16, presenting a significant optimization opportunity.

#### 3.3.1 The Prime Factor Problem

MLX's quantization implementation requires the quantized dimension to be divisible by the group size $g \in \{32, 64, 128\}$. The SigLIP intermediate dimension is:

$$d_{\text{inter}} = 4304 = 16 \times 269 \tag{4}$$

Since 269 is prime, $d_{\text{inter}}$ is not divisible by any supported group size:

$$4304 \div 32 = 134.5 \quad \text{(not integer)} \tag{5}$$

$$4304 \div 64 = 67.25 \quad \text{(not integer)} \tag{6}$$

$$4304 \div 128 = 33.625 \quad \text{(not integer)} \tag{7}$$

This prevents direct quantization of the `fc2` weight matrix $\mathbf{W}_{\text{fc2}} \in \mathbb{R}^{d_{\text{out}} \times 4304}$, where the input dimension (4304) must be group-aligned.

#### 3.3.2 Zero-Padding Solution

We pad the intermediate dimension to the next group-aligned value:

$$d'_{\text{inter}} = \lceil 4304/64 \rceil \times 64 = 68 \times 64 = 4352 \tag{8}$$

The padding procedure for each vision layer $\ell \in \{0, \ldots, 26\}$ is:

1. **fc1 expansion**: Dequantize $\mathbf{W}^{(\ell)}_{\text{fc1}}$ from 4-bit to bfloat16, pad from $\mathbb{R}^{4304 \times d_{\text{in}}}$ to $\mathbb{R}^{4352 \times d_{\text{in}}}$ by appending 48 zero rows, then requantize to 4-bit.

2. **fc1 bias**: Pad $\mathbf{b}^{(\ell)}_{\text{fc1}}$ from $\mathbb{R}^{4304}$ to $\mathbb{R}^{4352}$ by appending 48 zeros.

3. **fc2 quantization**: Pad $\mathbf{W}^{(\ell)}_{\text{fc2}}$ from $\mathbb{R}^{d_{\text{out}} \times 4304}$ to $\mathbb{R}^{d_{\text{out}} \times 4352}$ by appending 48 zero columns, then quantize from bfloat16 to 4-bit with group size 64.

**Proposition 1** (Mathematical Equivalence)**.** *The zero-padding preserves the layer output exactly. For input* $\mathbf{x} \in \mathbb{R}^{d_{in}}$*:*

$$\mathbf{a} = \sigma(\mathbf{W}_{fc1}\mathbf{x} + \mathbf{b}_{fc1}) \in \mathbb{R}^{4304} \tag{9}$$

$$\mathbf{a}' = \sigma(\mathbf{W}'_{fc1}\mathbf{x} + \mathbf{b}'_{fc1}) = [\mathbf{a}; \mathbf{0}_{48}] \in \mathbb{R}^{4352} \tag{10}$$

$$\mathbf{W}'_{fc2}\mathbf{a}' = [\mathbf{W}_{fc2} \mid \mathbf{0}] \cdot [\mathbf{a}; \mathbf{0}_{48}] = \mathbf{W}_{fc2}\mathbf{a} \tag{11}$$

*where* $\sigma$ *denotes the GELU activation and the appended zero neurons contribute nothing to the dot product.*

**Savings.** Quantizing 27 layers of fc2 from bfloat16 to 4-bit saves approximately 191 MB. The padding overhead (48 extra neurons per layer, quantized) is less than 1 MB total.

### 3.4 Step 3: Text Layer Pruning

Transformer-based language models exhibit significant redundancy in their deeper layers, where representations become increasingly refined but with diminishing marginal contribution [Men et al., 2024]. We exploit this by removing the three deepest layers (indices 31, 32, 33) from the 34-layer text decoder, retaining layers 0–30.

#### 3.4.1 Layer Selection Rationale

The deepest layers in a transformer are responsible for final task-specific refinement, while earlier layers capture more fundamental linguistic features [Jawahar et al., 2019]. For a generative model, removing the deepest layers has the least impact on the model's core language understanding capabilities. Moreover, these layers are often the most redundant, as the representation has already converged by layer 30 in a 34-layer model.

#### 3.4.2 Procedure

For each removed layer index $j \in \{31, 32, 33\}$, we delete all associated weight tensors:

- Self-attention: $\mathbf{W}_q^{(j)}, \mathbf{W}_k^{(j)}, \mathbf{W}_v^{(j)}, \mathbf{W}_o^{(j)}$
- MLP: $\mathbf{W}_{\text{gate}}^{(j)}, \mathbf{W}_{\text{up}}^{(j)}, \mathbf{W}_{\text{down}}^{(j)}$
- Layer normalization: $\gamma_{\text{input}}^{(j)}, \gamma_{\text{post\_attn}}^{(j)}, \gamma_{\text{pre\_ff}}^{(j)}, \gamma_{\text{post\_ff}}^{(j)}$

The remaining layers are renumbered contiguously (0–30), and `num_hidden_layers` in the configuration is updated from 34 to 31. If a `per_layer_intermediate_sizes` array exists from a previous step, the corresponding entries are also removed.

**Savings.** Each text layer occupies approximately 53 MB (4-bit quantized). Removing three layers saves $3 \times 53 \approx 159$ MB.

### 3.5 Step 4: Resolution Reduction

The SigLIP vision encoder processes images at $896 \times 896$ pixels with $14 \times 14$ patches, producing a $64 \times 64 = 4096$ grid of patch embeddings. These are pooled via average pooling with a $4 \times 4$ kernel to produce 256 image tokens for the text decoder. The self-attention mechanism in the vision encoder has computational cost $\mathcal{O}(n^2 \cdot d)$ where $n$ is the number of patches, making it the dominant source of compute and thermal dissipation during image processing.

#### 3.5.1 Resolution Selection

We evaluate three candidate resolutions:

The 672 px resolution provides a $\sim$3.2$\times$ reduction in vision self-attention compute while maintaining sufficient spatial resolution for accurate image understanding. As we demonstrate in Section 5.4, the 448 px resolution is too aggressive and leads to degenerate generation behavior.

Table 1: Resolution candidates and their computational characteristics. Patch size is $14 \times 14$ throughout.

| Resolution | Patches | Image Tokens | Attention Cost | Relative Cost |
|---|---|---|---|---|
| $896 \times 896$ (original) | $64 \times 64 = 4096$ | 256 | $\mathcal{O}(4096^2 d)$ | $1.0\times$ |
| $672 \times 672$ (selected) | $48 \times 48 = 2304$ | 144 | $\mathcal{O}(2304^2 d)$ | $0.32\times$ |
| $448 \times 448$ (rejected) | $32 \times 32 = 1024$ | 64 | $\mathcal{O}(1024^2 d)$ | $0.06\times$ |

#### 3.5.2 Position Embedding Interpolation

The learned position embeddings $\mathbf{P} \in \mathbb{R}^{n_{\text{orig}} \times d_{\text{vis}}}$ must be adapted from the original $64 \times 64$ grid to the new $48 \times 48$ grid. We reshape the position embedding matrix into a spatial grid and apply 2D bilinear interpolation:

$$\mathbf{P}_{\text{grid}} = \text{reshape}(\mathbf{P}, [64, 64, d_{\text{vis}}]) \tag{12}$$

$$\mathbf{P}'_{\text{grid}} = \text{bilinear\_interpolate}(\mathbf{P}_{\text{grid}}, [48, 48]) \tag{13}$$

$$\mathbf{P}' = \text{reshape}(\mathbf{P}'_{\text{grid}}, [2304, d_{\text{vis}}]) \tag{14}$$

For quantized position embeddings, we first dequantize to full precision, perform the interpolation, and then requantize. The configuration is updated: `image_size`: 672, `mm_tokens_per_image`: 144.

**Savings.** Disk savings are minimal ($\sim$1 MB for the smaller position embedding matrix). The primary benefit is runtime: reduced activation memory, lower attention computation, and significantly reduced thermal output during image processing.

### 3.6 Step 5: MLP Neuron Pruning

#### 3.6.1 Activation Profiling

We perform activation profiling by running 20 forward passes with diverse text inputs through the model, recording the mean absolute activation magnitude $\bar{a}_i^{(\ell)}$ for each neuron $i$ in each layer $\ell$:

$$\bar{a}_i^{(\ell)} = \frac{1}{T} \sum_{t=1}^{T} |\sigma(\mathbf{W}_{\text{gate}}^{(\ell)} \mathbf{h}_t + \mathbf{b}_{\text{gate}}^{(\ell)})_i| \tag{15}$$

where $T$ is the total number of tokens across all forward passes and $\sigma$ denotes the `gelu_pytorch_tanh` activation.

#### 3.6.2 Dead Neuron Discovery

Profiling reveals a striking pattern: layers 14–30 contain 60–100% "dead" neurons with mean activation below a threshold $\tau = 0.5$. Layer 29 is the most extreme case, with nearly 100% dead neurons. In contrast, layers 0–13 exhibit healthy activation patterns with the majority of neurons contributing meaningfully to the computation.

*[Placeholder: Heatmap showing per-layer neuron activation magnitudes. Layers 14–30 show predominantly low activations (dark), while layers 0–13 show healthy activation patterns (bright).]*

Figure 2: Neuron activation heatmap across layers. Layers 14–30 exhibit 60–100% dead neurons at threshold $\tau = 0.5$. Layer 29 is nearly 100% dead. Early layers (0–13) show healthy activation patterns and are protected from pruning.

#### 3.6.3 Pruning Strategy

We define the pruning policy with three constraints:

1. **Layer protection**: Layers 0–13 are protected and never pruned, preserving the model's core feature extraction capability.
2. **Activation threshold**: For each unprotected layer $\ell$, neuron $i$ is a candidate for removal if $\bar{a}_i^{(\ell)} < \tau$.
3. **Maximum reduction**: The pruned intermediate dimension $d'_\ell$ must satisfy:

$$d'_\ell \geq (1 - r_{\max}) \cdot d_\ell \tag{16}$$

   where $r_{\max} = 0.25$ is the maximum reduction ratio and $d_\ell$ is the original intermediate dimension.
4. **Group alignment**: The pruned dimension is aligned to the quantization group size:

$$d'_\ell = \left\lfloor \frac{|\{i : \bar{a}_i^{(\ell)} \geq \tau\}|}{g} \right\rfloor \cdot g \tag{17}$$

   where $g = 64$ is the group size. If the aligned count exceeds the maximum reduction constraint, we adjust upward.

Formally, the retained neuron count for layer $\ell$ is:

$$d'_\ell = \max\left(\text{align}_g(|\{i : \bar{a}_i^{(\ell)} \geq \tau\}|),\ \lceil (1 - r_{\max}) \cdot d_\ell / g \rceil \cdot g\right) \tag{18}$$

The neurons with the highest activation magnitudes are retained (up to $d'_\ell$), and the corresponding rows/columns are extracted from the MLP weight matrices:

- $\mathbf{W}_{\text{gate}}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_h} \to \mathbb{R}^{d'_\ell \times d_h}$ (remove rows)
- $\mathbf{W}_{\text{up}}^{(\ell)} \in \mathbb{R}^{d_\ell \times d_h} \to \mathbb{R}^{d'_\ell \times d_h}$ (remove rows)
- $\mathbf{W}_{\text{down}}^{(\ell)} \in \mathbb{R}^{d_h \times d_\ell} \to \mathbb{R}^{d_h \times d'_\ell}$ (remove columns)

The per-layer intermediate sizes are recorded as `per_layer_intermediate_sizes` in the model configuration:

$$[\underbrace{10240, \ldots, 10240}_{14}, \underbrace{7680, \ldots, 7680}_{17}] \tag{19}$$

**Savings.** Pruning 25% of neurons from 17 layers saves approximately 188 MB. Each pruned layer reduces by $\frac{0.25 \times 10240}{10240} \times$ (layer MLP size) $\approx 11$ MB per layer.

### 3.7 Step 6: Weight Splitting

The final step partitions the single `model.safetensors` file into two components:

- **`language_model.safetensors`** (1.9 GB): All text decoder weights, including the compact embedding, layer normalization parameters, and the final norm.
- **`vision_model.safetensors`** (231 MB): All SigLIP vision encoder weights and the multimodal projector.

A `model.safetensors.index.json` file maps each weight tensor to its respective file, following the HuggingFace multi-file safetensors convention.

**Runtime Benefits.** For text-only inference, the application loads only `language_model.safetensors`, consuming ∼2.2 GB of peak memory. When the user sends an image for the first time, the vision weights are loaded on demand, increasing peak memory to ∼2.5 GB. This lazy loading strategy is particularly valuable on memory-constrained mobile devices where text-only conversations are the common case.

## 4 Implementation

### 4.1 MLX Swift Integration

We implement the compressed model inference in Apple's MLX Swift framework [Hannun et al., 2023, Apple, 2024]. Several modifications to the standard Gemma 3 implementation in `mlx-swift-lm` are required:

#### 4.1.1 Token Map Support

The token map $\mathbf{M}$ must be loaded and applied during both embedding lookup and output logit computation. We wrap the token map as an `MLXArrayBox`—a reference-type wrapper that hides the array from MLX's automatic module parameter reflection:

```
class MLXArrayBox {
    let value: MLXArray
    init(_ value: MLXArray) { self.value = value }
}
```

This prevents the token map from being treated as a learnable parameter during weight loading, while keeping it accessible for the embedding lookup:

```
func callAsFunction(_ x: MLXArray) -> MLXArray {
    let indices = tokenMap.value.gathered(x.flattened())
    return compactEmbedding(indices).reshaped(shape + [embeddingDimension])
}
```

#### 4.1.2 Per-Layer Intermediate Sizes

The pruned model uses different MLP intermediate dimensions across layers. We modify the `TransformerModel` initialization to read `per_layer_intermediate_sizes` from the configuration and pass the appropriate dimension to each `MLP` layer:

```
for i in 0..<config.numHiddenLayers {
    let intermediateSize = config.perLayerIntermediateSizes?[i]
                           ?? config.intermediateSize
    layers.append(TransformerLayer(config, intermediateSize))
}
```

#### 4.1.3 Split Weight Loading

The `model.safetensors.index.json` file directs the weight loader to the appropriate safetensors file for each tensor. MLX Swift's built-in multi-file loading handles this transparently, but we ensure that vision weights are only loaded when the `VisionModel` module is first accessed.

### 4.2 Inference Pipeline

The `gemma-cli` tool provides a command-line interface for inference:

```
gemma-cli --model /path/to/compressed-model \
          --prompt "Describe this image" \
          --image photo.jpg \
          --max-tokens 200 \
          --temperature 0.0
```

The pipeline supports streaming token generation with real-time performance statistics (tokens per second, peak memory usage). It is built using Swift's `ArgumentParser` framework and the `MLXVLM` library for vision-language model inference.

## 5 Experiments

### 5.1 Experimental Setup

**Hardware.** All experiments are conducted on an Apple Silicon Mac with unified memory architecture. We report peak memory usage as measured by MLX's memory profiling utilities.

**Models.** We evaluate three configurations:

- **Original**: Gemma 3 4B IT QAT 4-bit, unmodified (2.8 GB).
- **Lite** (Step 4): After vocabulary pruning, vision fc2 quantization, text layer pruning, and resolution reduction (2.3 GB). This variant does not include neuron pruning or weight splitting.
- **Mobile** (Step 7): The fully compressed model after all pipeline stages (2.1 GB).

**Evaluation.** We assess models on four axes: (1) text generation quality, (2) image understanding accuracy, (3) inference speed (prompt processing and token generation, in tokens per second), and (4) peak memory consumption.

Table 2: Text-only inference performance. Prompt and generation speeds are in tokens per second. Peak memory includes model weights, KV cache, and intermediate activations.

| Model | Disk Size | Prompt (t/s) | Generation (t/s) | Peak Memory |
|---|---|---|---|---|
| Original | 2.8 GB | 109 | 90 | 2910 MB |
| Lite (Step 4) | 2.3 GB | ∼120 | ∼110 | ∼2500 MB |
| Mobile (Step 7) | 2.1 GB | 120 | 110 | 2231 MB |

Table 3: Image understanding performance across compression stages. Quality is assessed by human evaluation of the generated description's accuracy and completeness.

| Model | Prompt (t/s) | Gen (t/s) | Peak Mem | Quality |
|---|---|---|---|---|
| Original (896 px) | 54 | 27 | ∼5500 MB | Excellent |
| Step 3 (896 px) | 73 | 61 | 4850 MB | Good |
| Mobile (672 px) | 184 | 104 | 4358 MB | Good |

### 5.2 Text-Only Performance

We evaluate text generation with the prompt "Hello, how are you?" using greedy decoding (`temperature`=0.0) and a maximum of 100 tokens.

As shown in Table 2, the fully compressed Mobile model achieves a 22% improvement in generation speed (110 vs. 90 t/s) and a 23% reduction in peak memory (2231 vs. 2910 MB). The speedup is attributed to the reduced model size (fewer parameters to load and compute through) and the 25% neuron reduction in layers 14–30. Prompt processing speed improves modestly from 109 to 120 t/s, as prompt processing is primarily memory-bandwidth bound.

### 5.3 Image Understanding Performance

We evaluate image understanding using a test photograph of a pizza with accompanying condiments. The prompt is "Describe this image in detail." with greedy decoding and a maximum of 200 tokens.

The results in Table 3 demonstrate dramatic improvements in image prompt processing speed: 184 t/s for the Mobile model vs. 54 t/s for the original, a 3.4× speedup. This is primarily due to the reduced patch count (2304 vs. 4096), which quadratically reduces the self-attention computation. Generation speed improves from 27 to 104 t/s (3.9×), reflecting the cumulative effect of text layer pruning and neuron removal. Peak memory during image processing drops from ∼5.5 GB to ∼4.4 GB, bringing multimodal inference within practical mobile deployment budgets.

The Mobile model correctly identifies the pizza, its toppings (cheese, herbs), the pizza box, and accompanying condiments (sauce, seasoning). While descriptions are slightly less detailed than the original model, the core understanding is preserved.

### 5.4 Failure Mode Analysis

A critical contribution of this work is the systematic identification of compression techniques that fail catastrophically for VLMs. We document three such failures:

Table 4: Failed compression experiments. These configurations were evaluated and rejected during pipeline development.

| Experiment | Savings | Result |
|---|---|---|
| 448 px resolution | runtime only | Token repetition loops: the model correctly begins describing the image but enters a degenerate cycle, repeating phrases indefinitely. |
| Remove vision layers 12–15 | 35 MB | Complete image understanding failure: the model misidentifies a pizza as "skin texture" and produces hallucinated descriptions unrelated to the image content. |
| 80K vocabulary (v1) | 261 MB | Generation quality collapse: missing BPE-merged tokens cause the model to produce fragmented, nonsensical text for common English sentences. |

#### 5.4.1 448px Resolution Failure

At 448×448 pixels, the vision encoder produces only $32 \times 32 = 1024$ patches, pooled to 64 image tokens. This represents a 16× reduction in vision self-attention compute compared to the original. However, the reduced spatial information is insufficient for coherent image understanding: the model correctly begins its description (e.g., recognizing a pizza) but quickly enters a token repetition loop, generating the same phrases cyclically. This suggests a critical minimum information threshold for the vision encoder to provide sufficient grounding for the language model.

We measured performance before detecting the failure: prompt processing at 86 t/s, generation at 26 t/s, and peak memory of 3.86 GB. While the memory reduction is attractive, the generation quality is unusable.

#### 5.4.2 Vision Layer Removal Failure

Removing four SigLIP vision layers (12–15) saves only 35 MB but completely destroys the vision encoder's representational capacity. The model misidentifies a pizza photograph as "skin texture," producing hallucinated descriptions entirely unrelated to the image content. This demonstrates that the 27-layer SigLIP encoder has minimal redundancy—unlike the text decoder, where three layers can be removed with acceptable quality loss. The asymmetry is likely because the vision encoder is a pre-trained SigLIP model fine-tuned end-to-end, whereas the text decoder has inherent redundancy in its deeper layers.

Notably, this variant showed the fastest prompt processing (154 t/s) and low memory (3.80 GB), making it superficially attractive if one only considers efficiency metrics.

#### 5.4.3 Vocabulary Under-Pruning Failure

The initial vocabulary pruning to 80K tokens (based on dictionary words and inflections, without the ASCII vocabulary scan) appeared successful in basic testing but failed on more diverse prompts.

Table 5: Cumulative compression across pipeline stages. Each step builds on the output of the previous step.

| Step | Operation | Disk Size | Savings | Cumulative |
|---|---|---|---|---|
| Baseline | Original QAT 4-bit | 2.8 GB | — | — |
| Step 1 | Vocabulary pruning | 2.54 GB | 170 MB | 170 MB |
| Step 2 | Vision fc2 quantization | 2.35 GB | 191 MB | 361 MB |
| Step 3 | Text layer pruning | 2.19 GB | 159 MB | 520 MB |
| Step 4 | Resolution reduction | 2.19 GB | ∼1 MB | 521 MB |
| Step 5 | Neuron pruning | 2.00 GB | 188 MB | 709 MB |
| Step 6 | Weight splitting | 2.10 GB* | — | 709 MB |

*Split total: 1.9 GB language + 231 MB vision = 2.1 GB. Slight increase due to index metadata.

The root cause is that BPE tokenization creates merged subword tokens that do not appear in any dictionary. For example, the token "`_I`" (the word "I" with a leading space) is a common BPE merge that maps to a single token ID. Without this token, the model must fall back to byte-level encoding, breaking the expected input distribution and causing generation to collapse.

Adding the ASCII vocabulary scan increased the retained token count from ∼80K to ∼144K, resolving all observed quality issues. This establishes that BPE token coverage, not dictionary coverage, is the binding constraint for vocabulary pruning.

### 5.5 Incremental Compression Analysis

Table 5 shows the cumulative effect of each compression stage. The three largest contributors are vision fc2 quantization (191 MB), neuron pruning (188 MB), and vocabulary pruning (170 MB). Resolution reduction contributes minimal disk savings but provides the largest runtime benefit through reduced activation memory and attention computation.

### 5.6 Quality Analysis

**Text Generation.** The compressed model produces coherent, fluent English text. However, we observe minor degradation in contraction handling: the model occasionally generates "I's" instead of "I'm," likely due to the combined effect of layer pruning and neuron removal altering the model's learned distribution over short common phrases. This is a known consequence of structured pruning without subsequent fine-tuning.

**Image Understanding.** At 672 px, the model retains the ability to correctly identify objects, scenes, and attributes in photographs. Descriptions are somewhat less detailed than the original 896 px model—for example, mentioning fewer specific toppings on a pizza—but the core understanding is preserved. The model does not hallucinate or misidentify objects, in stark contrast to the 448 px and vision-layer-pruned variants.

# 6 Discussion

## 6.1 Compression-Quality Trade-offs

Our pipeline achieves a 25% reduction in disk size (2.8→2.1 GB) and a 23% reduction in text-only peak memory (2.9→2.2 GB) while maintaining functional text and image understanding. However, the quality-compression frontier is not smooth: small additional compression (*e.g.*, 35 MB from vision layer removal) can cause catastrophic failure, while large compressions (*e.g.*, 191 MB from fc2 quantization) can be lossless.

This asymmetry arises from the heterogeneous sensitivity of VLM components. The text decoder tolerates significant pruning (3 layers, 25% of neurons in 17 layers) because its 34-layer depth provides substantial redundancy. The vision encoder, with its 27 tightly-coupled SigLIP layers fine-tuned for visual representation, tolerates almost no structural modification.

## 6.2 The BPE Coverage Threshold

Our finding that vocabulary pruning requires ∼144K tokens (not ∼80K) highlights a subtle but critical aspect of subword tokenization. Dictionary-based token selection misses BPE-merged forms that are essential for the model's learned input distribution. The model's embedding space was trained with specific BPE tokens as atomic units; removing these tokens and forcing byte-level fallback fundamentally disrupts the expected input representation.

This suggests a general principle for vocabulary pruning: *the retained vocabulary must include all BPE merges that occur in the target language's character set, not just the tokens corresponding to dictionary words.* For English, this means retaining all ASCII-printable BPE merges, which approximately doubles the vocabulary requirement compared to a dictionary-only approach.

## 6.3 Resolution as Thermal Management

An unexpected finding is that resolution reduction is the single most effective technique for reducing thermal output during image processing, despite contributing negligible disk savings. The ∼3× reduction in vision self-attention compute translates directly to reduced GPU utilization and heat generation. On mobile devices where thermal throttling limits sustained performance, this is arguably more important than raw speed improvements.

The quadratic scaling of self-attention with patch count ($\mathcal{O}(n^2)$) means that modest resolution reductions yield large computational savings: a 25% reduction in linear resolution (896→672) reduces attention cost by ∼68%. This is significantly more efficient than reducing the number of attention layers, which scales linearly.

## 6.4 The Vision Encoder Sensitivity Puzzle

The extreme sensitivity of the SigLIP vision encoder to layer removal ($-4$ layers → complete failure) contrasts with the text decoder's robustness ($-3$ layers → minor quality loss). We hypothesize three contributing factors:

1. **Pre-training regime**: SigLIP is pre-trained with contrastive learning on image-text pairs, creating tightly interdependent layer representations. The text decoder, trained with autoregressive language modeling, develops more modular, hierarchical representations.

2. **Information bottleneck**: The vision encoder must compress a high-dimensional image ($672 \times 672 \times 3 \approx 1.35$M values) into 144 tokens, requiring every layer to contribute to this lossy compression. The text decoder operates on a sequence that is already in the model's native representation space.

3. **Absolute layer count**: 27 layers is already compact for a vision encoder processing images of this complexity. The model has less redundancy to spare compared to the 34-layer text decoder.

### 6.5 Limitations

Our evaluation has several limitations. First, we evaluate on a small set of qualitative examples rather than standardized VLM benchmarks (VQAv2, MMLU, etc.), as our focus is on deployment feasibility rather than leaderboard performance. Second, the neuron pruning threshold ($\tau = 0.5$) and maximum reduction ratio ($r_{\max} = 0.25$) were selected based on limited experimentation; a more thorough hyperparameter search might identify better operating points. Third, we do not apply knowledge distillation or fine-tuning after compression, which could recover some of the observed quality degradation. Finally, while we report memory and speed measurements on Apple Silicon, we have not yet validated on actual iPhone/iPad devices, where memory pressure from background processes and thermal throttling may affect results.

## 7 Conclusion and Future Work

We have presented GEMMA-PRUNE, a multi-stage compression pipeline that reduces the Gemma 3 4B IT QAT vision-language model from 2.8 GB to 2.1 GB while preserving both text generation and image understanding capabilities. The pipeline combines six complementary techniques—vocabulary pruning, vision encoder quantization with dimension padding, text layer removal, resolution reduction, dead neuron pruning, and weight splitting—each targeting a distinct source of redundancy.

Key results include:

- **25% disk reduction**: 2.8 GB → 2.1 GB (709 MB total savings).
- **23% memory reduction**: 2910 MB → 2231 MB for text-only inference.
- **3.4× faster image processing**: 184 t/s vs. 54 t/s prompt processing.
- **22% faster generation**: 110 t/s vs. 90 t/s for text-only inference.
- **∼3× vision attention reduction**: through 896→672 px resolution change.

Equally important, we document three critical failure modes: (1) vision layer removal destroys image understanding with minimal savings, (2) 448 px resolution induces token repetition loops, and (3) vocabulary pruning below ∼144K tokens causes generation collapse due to missing BPE merges. These findings provide practical guidelines for future VLM compression research.

**Future Work.** Several directions remain for exploration:

1. **Knowledge distillation**: Fine-tuning the compressed model with the original model as teacher could recover quality degradation, particularly for text contractions and detailed image descriptions.

2. **Dynamic resolution**: Adapting resolution based on image content (high resolution for detailed scenes, low resolution for simple images) could further optimize the compute-quality trade-off.

3. **Device-specific profiling**: Measuring actual thermal behavior and sustained throughput on iPhone and iPad devices under realistic usage patterns.

4. **Attention pruning**: Exploring structured pruning of attention heads (complementary to neuron pruning) in both the text decoder and vision encoder.

5. **Speculative decoding**: Combining the compressed model with a smaller draft model for speculative decoding could further improve generation speed.

6. **Generalization**: Applying the pipeline to other VLMs (LLaVA, Phi-3 Vision, Qwen-VL) to validate the generality of our findings.

## References


Gemma Team, Riviere, M., Pathak, S., Sessa, P. G., Hardin, C., Bhupatiraju, S., Hussenot, L., Mesnard, T., Shahriari, B., Ramé, A., Ferret, J., Liu, P., Tafti, P., Precup, D., and others. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Hannun, A., Baradad, M., Shelekhov, I., Filipi, A., and others. MLX: Efficient machine learning on Apple silicon. *Apple ML Research*, 2023. URL https://github.com/ml-explore/mlx.

Apple. MLX Swift: Swift API for MLX. https://github.com/ml-explore/mlx-swift, 2024.

Zhai, X., Mustafa, B., Kolesnikov, A., and Beyer, L. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11975–11986, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

Nagel, M., Amjad, R. A., van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? Adaptive rounding for post-training quantization. In *International Conference on Machine Learning (ICML)*, 2022.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: Activation-aware weight quantization for efficient LLM compression and acceleration. In *Machine Learning and Systems (MLSys)*, 2024.

Ma, X., Fang, G., and Wang, X. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Men, X., Xu, M., Zhang, Q., Wang, B., Lin, H., Lu, Y., Han, X., and Chen, W. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*, 2024.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *5th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC2)*, 2019.

Fang, Z., Wang, J., Wang, L., Yang, L., Jiao, B., Wang, X., and Shou, M. Z. Compressing vision-language models via efficient knowledge distillation. *arXiv preprint arXiv:2403.09016*, 2024.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. Visual instruction tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Bikowski, M., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. Flamingo: a visual language model for few-shot learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shang, Y., He, K., Yuan, Z., Wang, Z., and Wang, Z. LLaVA-PruMerge: Adaptive token pruning and merging for efficient large multimodal models. *arXiv preprint arXiv:2403.15388*, 2024.

Cai, W., Peng, Z., Beyer, L., Steiner, A., Zhai, X., and Kolesnikov, A. FlexiViT: One model for all patch sizes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Apple. Core ML: Integrate machine learning models into your app. https://developer.apple.com/documentation/coreml, 2023.

Gerganov, G. llama.cpp. https://github.com/ggerganov/llama.cpp, 2023.

Kudo, T. and Richardson, J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, 2018.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1715–1725, 2016.

Zheng, Q., Wang, X., Li, Y., and Tan, Y. Trimming the fat: Efficient vocabulary pruning for pre-trained language models. *arXiv preprint arXiv:2404.14573*, 2024.

Jawahar, G., Sagot, B., and Seddah, D. What does BERT look at? An analysis of BERT's attention. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, 2019.

Table 6: Original Gemma 3 4B IT QAT 4-bit configuration parameters.

| | |
|---|---|
| **Text Decoder** | |
| Vocabulary size | 262,208 |
| Hidden size | 2560 |
| Intermediate size (MLP) | 10,240 |
| Number of layers | 34 |
| Attention heads | 8 (GQA) |
| Key-value heads | 4 |
| Head dimension | 256 |
| Activation function | `gelu_pytorch_tanh` |
| Sliding window | 1024 |
| Sliding window pattern | 6 |
| Tied embeddings | Yes |
| Quantization | 4-bit, group_size=64 |
| **Vision Encoder (SigLIP)** | |
| Hidden size | 1152 |
| Intermediate size | 4304 |
| Number of layers | 27 |
| Attention heads | 16 |
| Image size | $896 \times 896$ |
| Patch size | $14 \times 14$ |
| Image tokens per image | 256 |
| **Special Tokens** | |
| BOI token index | 255999 |
| EOI token index | 256000 |
| Image token index | 262144 |
| EOS token IDs | [1, 106] |

# A Model Configuration Details

## A.1 Original Model Configuration

## A.2 Compressed Model Configuration

# B Compression Pipeline Pseudocode

# C Detailed Benchmark Results by Pipeline Stage

Table 7: Compressed Gemma-Prune model configuration.

| | |
|---|---|
| **Text Decoder** | |
| Vocabulary size (logical) | 262,208 |
| Compact embedding size | 144,257 |
| Token map | int32[262208] $\rightarrow$ compact index |
| Number of layers | 31 (removed 31, 32, 33) |
| Per-layer MLP sizes | $[10240] \times 14 + [7680] \times 17$ |
| **Vision Encoder** | |
| Intermediate size | 4352 (padded from 4304) |
| Number of layers | 27 (all retained) |
| Image size | $672 \times 672$ |
| Image tokens per image | 144 |
| fc2 quantization | 4-bit, group_size=64 |
| **Weight Files** | |
| `language_model.safetensors` | 1.9 GB |
| `vision_model.safetensors` | 231 MB |
| Total | 2.1 GB |

Table 8: Image understanding benchmarks at each compression stage using a test photograph of a pizza. Prompt: "Describe this image in detail." with `temperature`=0.0, `max_tokens`=200.

| Configuration | Prompt (t/s) | Gen (t/s) | Peak Mem (MB) | Quality |
|---|---|---|---|---|
| Original (896 px) | 54.3 | 26.6 | 5540 | Excellent |
| Step 3 (896 px) | 72.7 | 61.3 | 4850 | Good |
| Step 4 / 672 px | 122.7 | 65.7 | 4590 | Good |
| Step 4 / 448 px | 86.1 | 26.0 | 3860 | Failed (repetition) |
| Step 5 / $-4$ vis. layers | 153.9 | 63.5 | 3800 | Failed (hallucination) |
| Mobile (Step 7, 672 px) | 184 | 104 | 4358 | Good |

**Algorithm 1** Gemma-Prune Compression Pipeline

**Require:** Original model $\mathcal{M}_0$ (Gemma 3 4B IT QAT 4-bit)
**Ensure:** Compressed model $\mathcal{M}_6$

**Step 1: Vocabulary Pruning**
$\mathcal{T} \leftarrow$ CollectTokens(dictionary, ASCII, special, byte_fallback)
$\mathbf{E}' \leftarrow$ ExtractEmbeddings($\mathcal{M}_0$, $\mathcal{T}$)
$\mathbf{M} \leftarrow$ BuildTokenMap($\mathcal{T}$, $V_{\text{orig}}$)
$\mathcal{M}_1 \leftarrow$ ReplaceEmbedding($\mathcal{M}_0$, $\mathbf{E}'$, $\mathbf{M}$)
**Step 2: Vision fc2 Quantization**
**for** $\ell = 0$ to 26 **do**
  $\mathbf{W}_{\text{fc1}}^{(\ell)} \leftarrow$ PadRows(Dequant($\mathbf{W}_{\text{fc1}}^{(\ell)}$), 4352)
  $\mathbf{W}_{\text{fc2}}^{(\ell)} \leftarrow$ Quantize(PadCols($\mathbf{W}_{\text{fc2}}^{(\ell)}$, 4352), 4-bit, $g$=64)
**end for**
$\mathcal{M}_2$.config.vision.intermediate_size $\leftarrow$ 4352
**Step 3: Text Layer Pruning**
$\mathcal{M}_3 \leftarrow$ RemoveLayers($\mathcal{M}_2$, text, $\{31, 32, 33\}$)
$\mathcal{M}_3$.config.num_hidden_layers $\leftarrow$ 31
**Step 4: Resolution Reduction**
$\mathbf{P}' \leftarrow$ BilinearInterp2D($\mathbf{P}$, 64×64 $\rightarrow$ 48×48)
$\mathcal{M}_4$.config.image_size $\leftarrow$ 672
**Step 5: Neuron Pruning**
$\bar{\mathbf{a}} \leftarrow$ ProfileActivations($\mathcal{M}_4$, $T$=20)
**for** $\ell = 14$ to 30 **do**
  $\mathcal{S}_\ell \leftarrow$ TopK($\bar{\mathbf{a}}^{(\ell)}$, $d'_\ell$) {Retain top neurons}
  PruneMLP($\mathcal{M}_5^{(\ell)}$, $\mathcal{S}_\ell$)
**end for**
**Step 6: Weight Splitting**
SplitWeights($\mathcal{M}_5$) $\rightarrow$ {language_model.safetensors, vision_model.safetensors}
$\mathcal{M}_6 \leftarrow$ UpdateIndex($\mathcal{M}_5$)
**return** $\mathcal{M}_6$