EvoComp: Learning Visual Token Compression for Multimodal Large Language Models via Semantic-Guided Evolutionary Labeling

Supplementary Material

A. Training Details

Evolutionary Labeling. To train the visual token compressor, we generate supervision labels using the evolutionary algorithm for LLaVA-1.5-7B[1], LLaVA-NeXT-7B[2] and Qwen2.5-VL-7B[3] models, respectively. The training dataset consists of LLaVA [29], VQAv2 [13], GQA [17], OKVQA [37], OCRVQA [38], A-OKVQA [44], TextVQA [46], and ScienceQA [36]. For each image-text pair, we apply the following configuration during the search: the population size q is 48, the number of parents p is 12, and the number of iterations L is 10. All randomness involved in the evolutionary algorithm is controlled using the Python (3.10) `random` package. For LLaVA-1.5-7B, we restrict the search space to the 64 largest token subsets (in terms of the number of visual tokens per subset) for each sample. During evolution, only these 64 subsets are considered for mask optimization, and tokens in all other subsets are automatically dropped, that is, masked as 0. This ensures that each resulting mask contains exactly 64 retained tokens, promoting consistency and controllability across the samples. Similarly, for Qwen2.5-VL-7B, we restrict the search space to the 11.1% largest token subsets for each sample. This search strategy produces high-quality, diverse, and task-aligned binary labels, which are used to supervise compressor training.

Compressor Training. We initialize the compressor using the parameters of the first transformer layer of the LLM in the target MLLM. This initialization provides a stable starting point and facilitates a faster convergence. The number of parameters in the compressor for LLaVA-1.5-7B and LLaVA-NeXT-7B is 202M, while for Qwen2.5-VL-7B it is 233M. Our models are implemented using PyTorch 2.1.2 and trained on four NVIDIA A100-80G GPUs. The optimizer is Adam, with an initial learning rate of 0.003 and a cosine learning rate decay schedule. The batch size is set to 256 for training the compressor used with LLaVA-1.5-7B, 32 for that used with LLaVA-NeXT-7B, and 48 for that used with Qwen2.5-VL-7B. To balance the GHM and cosine similarity losses, the loss weight coefficient α is set to 1. We adopt the unit region variant [21] of the GHM loss during training, with the number of unit regions set to 100, 150 and 150 for LLaVA-1.5-7B, LLaVA-NeXT-7B and Qwen2.5-VL-7B, respectively. The total number of training epochs is 30, and the best checkpoint is selected based on the performance on the TextVQA validation set. This training configuration enables efficient learning of token importance and diversity, resulting in a lightweight yet effective visual token compressor.

[1] https://huggingface.co/liuhaotian/llava-v1.5-7b

[2] https://huggingface.co/liuhaotian/llava-v1.6-vicuna-7b

[3] https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct

B. Evaluation Datasets

GQA. GQA [17] is a large-scale dataset designed to evaluate scene understanding through the task of visual question answering. It addresses key limitations of earlier benchmarks by reducing language priors, encouraging compositional reasoning, and enabling fine-grained diagnostic analysis. GQA contains questions grounded in real-world images, each accompanied by a scene graph that represents objects, attributes, and relationships. The dataset emphasizes multistep inference, spatial reasoning, and semantic compositionality, making it significantly more challenging than prior VQA datasets.

MMBench/MMBench-CN. MMBench [35] is a multimodal large-scale benchmark comprising multiple-choice questions collected from public datasets and the Internet. It evaluates models in 20 fine-grained ability dimensions, hierarchically organized into three levels (L-1 to L-3), spanning both perception and reasoning capabilities. Unlike task-specific or subjective benchmarks, MMBench provides objective, reproducible evaluation across diverse multimodal skills, offering detailed diagnostic feedback that aids in the development and analysis of MLLMs. MMBench-CN is the Chinese version of this dataset, allowing for a comparison of the model's performance in English and Chinese with the same set of images and questions.

POPE. POPE [25] is a diagnostic benchmark specifically designed to evaluate object hallucination in MLLMs. It systematically constructs image-text pairs to test whether a model generates object references inconsistent with the visual input. The dataset consists of three types of samples: positive (where the object mentioned is clearly present in the image), negative (where the object is absent) and neutral (where the presence of the object is ambiguous). By controlling object positioning and textual cues, POPE enables a fine-grained analysis of a model's robustness, visual grounding ability, and sensitivity to language priors.

VQAv2. VQAv2 [13] serves as a benchmark to evaluate joint vision-language understanding through open-ended visual question answering. The dataset includes over 1M natural-language questions grounded on 265,016 images from MS-COCO and abstract scenes, with each question annotated by 10 human responses. The questions span bi-

nary (yes/no), counting, and open-ended types, and the evaluation is based on consensus among the annotators.

VizWiz. VizWiz [15] is a VQA dataset specifically designed to support the development of assistive AI technologies for blind people. It comprises images captured by blind users, each paired with a spoken question they recorded about the image. These visual questions are further annotated with 10 crowd-sourced answers. Unlike conventional VQA datasets, VizWiz introduces unique challenges such as poor image quality and unanswerable questions. Consequently, the benchmark includes two tasks: predicting an answer to a visual question and determining whether the question is answerable. VizWiz provides a realistic and impactful setting for developing inclusive, accessibility-focused AI systems.

C. On-Device NPU Deployment Details

All on-device experiments in this work are conducted using LLaVA-1.5-7B alongside its corresponding compressor, whose compact parameter size makes it well-suited for mobile deployment. The smartphone is equipped with 16GB of LPDDR5X memory, 1TB of UFS 4.0 storage, and the MT6991 chipset, and comes with Android 15 installed. Although the smartphone used here is among the latest flagship products at the time of writing so that we can report the state-of-the-art performance values, we do not mean to restrict our proposed scheme to be applicable only to a few devices or chipsets. Instead, we argue that the proposed method should work on a wide spectrum of platforms, whether on cloud or at edge, on high-end smartphones and tablets, or on low-cost embedded devices. In the following, we briefly expand on the executive details.

C.1. Model Preparation

Initially, the MLLM and its accompanying compressor follow the publicly available format of Hugging Face. Note that the precision at this stage is `bfloat16`.

C.2. Model Quantization

As mentioned above, an MLLM can essentially be viewed as a combination of a ViT (along with the alignment module) serving as the visual encoder, and an LLM functioning as the text generator. Next, we discuss the strategies employed for each component, including the compressor.

LLM. Following general practice, PTQ is performed on a CUDA server. The LLM part is quantized to integer precisions as much as possible to allow for accelerated inference on the mobile NPU, as it takes up the majority of the total parameter count. We set the quantization target for the LLM part at `int4` weights, `int16` activations, and `int8` KV caches (`W4A16KV8`) as usual.

ViT and Compressor. Per recommendation of the chip documentation, both ViT and compressor are left with `float`. This is necessary to maintain accuracy while performing inference with acceptable efficiency.

C.3. Shape Fixing

Unlike cloud-based inference, the mobile NPU lacks support for dynamic input shapes, batching, and other flexible execution features. Instead, it requires fixed-shape inputs and outputs. This introduces additional challenges in model conversion, since an MLLM inference is no longer a single forward pass, as is the case with conventional models like CNNs. To address this, we adopt a strategy that involves producing multiple fixed-shape model variants, as detailed below.

ViT. This part is responsible for producing visual embedding matrices, so we need to know the target resolution to which each input image is resized. Once fixed, this is logically equivalent to a fixed number of visual tokens.

LLM. The LLM component operates in a two-stage pipeline consisting of a prefilling stage and a decoding stage. The two stages run at different levels of parallelism, which essentially translates to different sequence lengths. For mobile deployment, it is common to adopt a sequence length of 128 tokens[4] (sometimes referred to as the `AR128` mode where AR is short for "**a**uto**r**egressive") in the prefilling stage and of 1 token (`AR1`) in the decoding stage. During inference, the prefilling stage works by processing the input embeddings broken into longer subsequences so that the computational power is fully utilized to quickly fill up the KV caches within the context window, while the decoding stage repeatedly generates one token per forward pass based on the most updated context until the end-of-sequence (EOS) token is produced. Another deployment constraint involves fixing the context length. We find that 1,024 and 256 tokens appear to be adequate to hold any complete conversation (i.e., inclusive of both input and output tokens) for the tasks evaluated before and after compression, respectively. Based on these settings, we construct four fixed-shape LLM variants in total, denoted `128t1024c`, `1t1024c`, `128t256c` and `1t256c` (where `t` and `c` stand for the sequence length and context length, respectively), which are coupled as appropriate in the evaluations.

Compressor. This part lies between the ViT alignment module and the LLM, so its input sequence length is simply aligned to that of the LLM.

C.4. Compilation

Offline Compilation. To enable execution on the NPU, the models still need to be converted to bare-metal commands that instruct the logic gates and registers cycle by cycle.

[4] This basically results from the maximum level of parallelism in the chip design. Left padding would exist if the number of tokens to be encoded in a single inference is not a multiple of 128.

This compilation process is performed offline on a host PC, generating native-format binaries that can be efficiently deployed and reused across inference runs.

Optimizations (if applicable). For the LLM part, it comes to our attention that all four fixed-shape LLMs differ only in their input and output shapes. In fact, all of their weight parameters are shared between them but stored twice per context length (or totally four times in our case). This is a huge waste of the scarce storage space on the device. Hence, we have used a vendor-supplied utility tool to extract the shared weights among the models.

C.5. Inference on Device

We execute the compiled (and optimized) models on the smartphone using an inference framework adapted from llama.cpp [11] to support the on-chip NPU. To ensure reproducibility and fair performance comparisons, the CPU, memory and NPU of the smartphone are set to their maximum operating frequencies prior to each experiment. For each input instance, we first obtain the original visual embeddings via the ViT and the embedded text tokens, and then pass them through the compressor for per-token retention decisions. After that, we only keep the preserved visual embeddings, combine them with the text embeddings, and send them together into the LLM to get the final output.

D. Visualization of Token Selection

We further conduct a qualitative analysis to examine the effectiveness of our compressor in selecting informative and non-redundant visual tokens. For each image–question pair, we visualize the spatial locations of the visual tokens retained by the compressor of LLaVA-1.5-7B. Figure 5 presents examples where we retain 64, 128, and 192 tokens for a series of yes/no questions focused on object existence. The preserved tokens are strongly aligned with the objects mentioned in the questions. As the number of retained tokens increases, more contextual details are included, but the most critical regions are preserved in most cases, even with as few as 64 tokens. In Figure 6, we further examine open-ended questions that require more diverse reasoning types, including text recognition, attribute identification, activity understanding, object counting, and scene description. Here, we visualize only the 64-token setting. Despite the aggressive compression, the retained tokens consistently cover semantically salient regions that are relevant to answering the questions.

These visualization results demonstrate that our method can effectively identify and preserve question-relevant visual information while discarding redundant or irrelevant tokens. The strong alignment between the selected tokens and the question highlights the compressor's ability to perform content-aware filtering. This qualitative evidence complements our quantitative results, showing that the EvoComp approach not only maintains task performance, but also offers semantically grounded token selection.

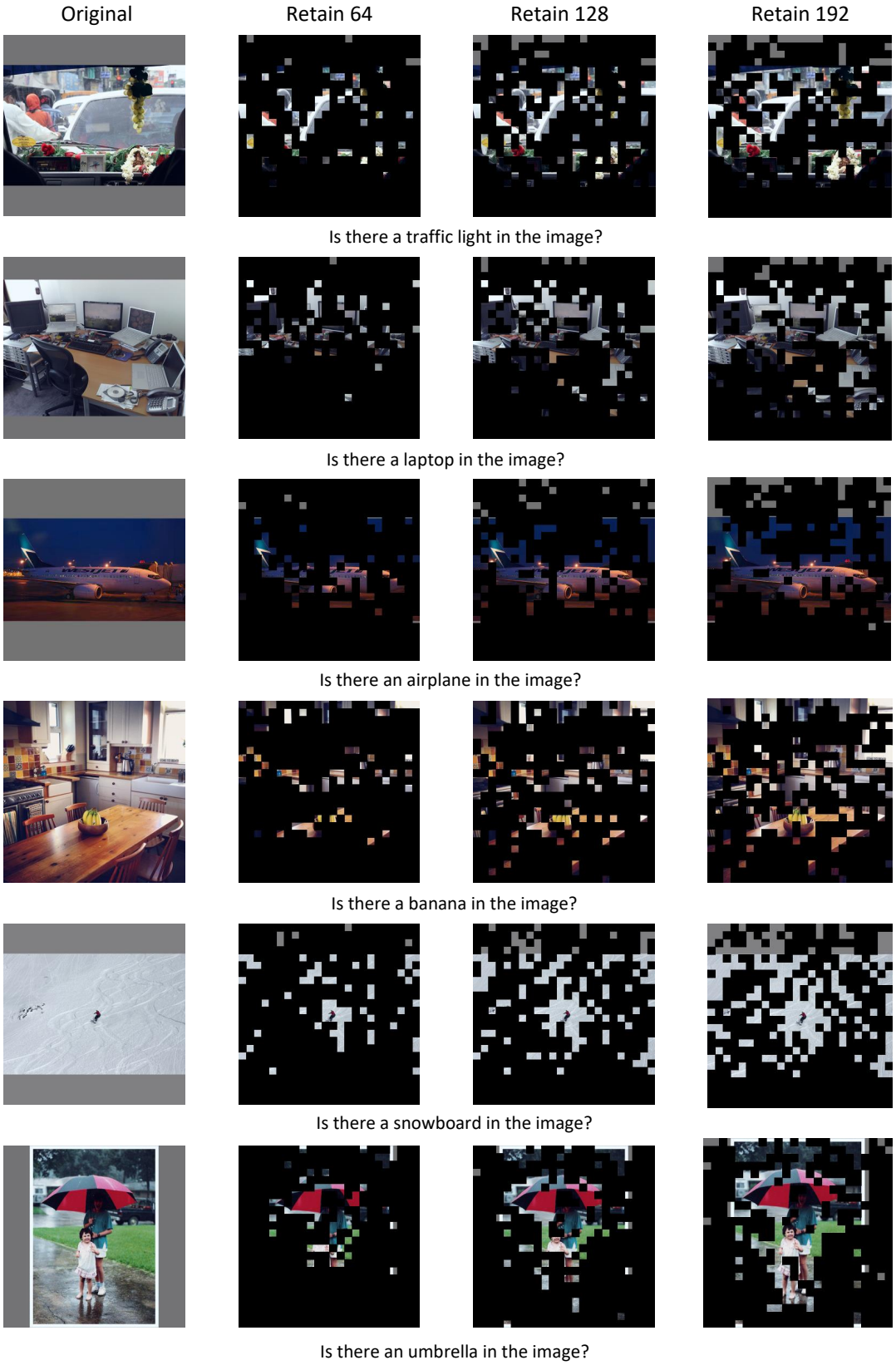

Figure 5. Visualization of tokens retained by EvoComp under different compression levels for yes/no questions.

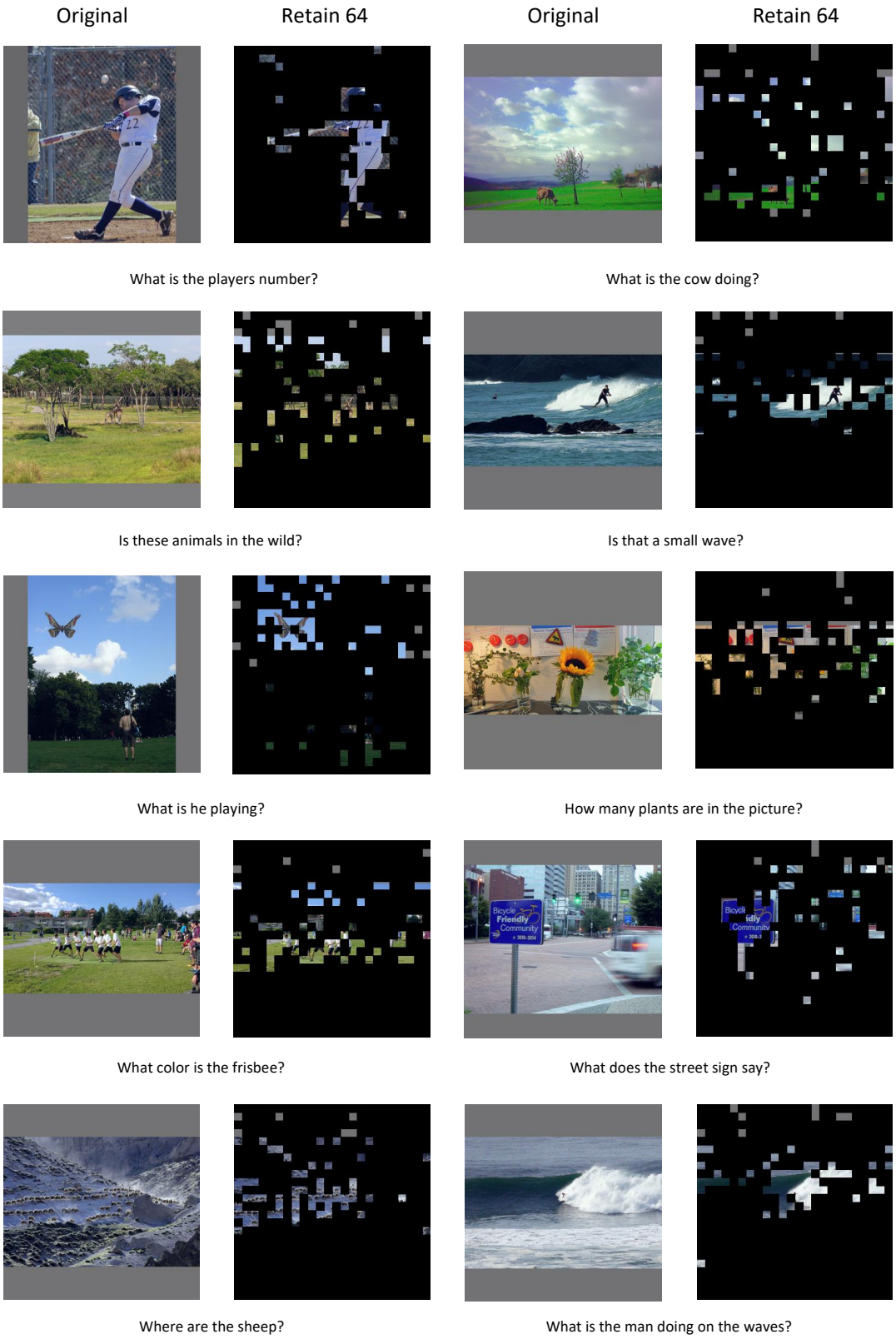

Figure 6. Visualization of tokens retained by EvoComp for open-ended questions.