BLIP Vision Explanation Studio: An Interactive Image Captioning and Simple VQA System

Sangeen Khan (2025A8017729003)
University of Chinese Academy of Sciences, Beijing, China
Email: sangeenkhan2662@mails.ucas.ac.cn

***Abstract*—This report documents the design and implementation of *BLIP Vision Explanation Studio*, an interactive web application for image captioning and simple visual question answering (VQA). The system is implemented in Python using PyTorch, Hugging Face Transformers, and Gradio. It wraps a pretrained BLIP image captioning model in a professionally styled dashboard interface that lets users upload an image, optionally ask a question, and receive a concise caption and answer generated directly by BLIP. Due to limited computational resources, a planned Ollama based local LLM backend could not be deployed, so a lighter BLIP based pipeline was adopted instead.**

I. Algorithm Performance Testing Methodology

A. Standardized Testing Environment

The project is implemented in a single Python script or notebook that first installs the required packages:

- `transformers` and `accelerate` for the BLIP model and processor,
- `torch` and `torchvision` as the deep learning backend,
- `pillow` for image handling via `PIL.Image`,
- `gradio` for building the web based interface.

The compute device is selected dynamically with a simple rule: use the GPU (`"cuda"`) if a CUDA device is available, otherwise use the CPU (`"cpu"`). This rule is implemented with a single conditional expression in Python, so the same script runs unchanged on different hardware.

The BLIP processor and model are instantiated using the standard Hugging Face `from_pretrained` API. A vision–language processor handles image preprocessing and text tokenization, and an encoder–decoder captioning model performs generation. Both objects are moved to the selected device; the model uses half precision on GPU and single precision on CPU, and runs in evaluation mode so that gradients are disabled during inference. This description avoids long inline code fragments and fits comfortably within a single IEEE column.

B. Resource Constraints and Ollama Limitation

The initial design goal was to use a local Ollama based LLM backend, so that all vision and language reasoning would run on device. This approach requires a large amount of GPU memory, CPU resources, and disk space for the Ollama runtime and model weights. In the available environment (limited Colab tier and modest local hardware), these requirements could not be satisfied: the models could not be loaded reliably and long inference runs were unstable.

Because of this, the system adopts a lighter alternative: a single BLIP captioning model loaded via Hugging Face Transformers, combined with a compact Gradio dashboard. This keeps memory and compute demands within the available budget while still providing useful image captions and simple VQA style answers.

C. Test Data Design

The application handles arbitrary user supplied images and optional questions via the Gradio UI:

- an image component configured to return `PIL.Image` objects, and
- a textbox component that captures an optional question string.

Inside the `analyze_image` function, the input image is validated. If it is missing, a warning message is returned. If it is not already a `PIL.Image`, it is converted from a NumPy array. Very large images are downscaled by limiting the larger side to 512 pixels before being sent to BLIP. This bounds the spatial resolution and prevents excessive processing time and memory use.

Thus the test data consists of natural images (scenes, objects, people, UI screenshots) and short natural language questions, or no question for caption only mode.

D. Performance Metrics

The code is structured so that the following metrics can be assessed:

- response latency from clicking *Analyze Image* to seeing the result,
- qualitative device behavior on CPU versus GPU,
- caption quality in terms of informativeness and fluency,
- answer quality in terms of relevance and visual grounding,
- robustness to missing inputs and large images,
- user experience of the Gradio interface and custom styling.

E. Testing Procedure

The testing procedure is:

1) install dependencies with `pip`,
2) import PyTorch, BLIP classes, `PIL.Image`, and Gradio,
3) detect the device, load the BLIP processor and model, and switch the model to evaluation mode,
4) define `get_blip_caption` and `analyze_image`,
5) define the Gradio Blocks UI and apply the custom CSS,
6) launch the interface with `demo.launch(share = True)`,
7) for each test image, upload the image, optionally type a question, click *Analyze Image*, and inspect the caption and answer.

II. Reference Case: BLIP Captioning and Simple VQA

A. Tested Algorithms

The project uses one pretrained vision–language model, BLIP, in two modes:

- **caption only mode**, implemented by `get_blip_caption`, which encodes the image, calls the BLIP `generate` method with beam search, and decodes a short caption; and
- **caption plus question mode**, implemented in `analyze_image`, which builds a simple question prompt, encodes image and text together, and calls `generate` again to produce an answer.

In both modes, the model runs in inference only configuration with gradients disabled.

B. Performance Test Results

1) Execution Time Comparison: Latency is controlled by three main choices:

- downscaling images larger than 512 pixels on the long side,
- using half precision on GPU and single precision on CPU, and
- always running under `torch.no_grad()`.

Caption only and caption plus question modes both require a single encoder–decoder pass with beam search; the question only affects the conditioning, not the overall complexity.

2) Performance Ranking: Qualitatively, the expected performance ranking is:

- GPU with downscaling: most responsive,
- GPU without downscaling: slightly slower on very large images,
- CPU with downscaling: suitable for light use and teaching,
- CPU without downscaling: slowest, mainly for offline exploration.

Algorithm 1 Conceptual BLIP Text Generation

Require: image `I`, optional question `q`

```
encode image and question into model inputs
run beam search text generation on the BLIP model
select the best generated text sequence
return generated text
```

C. Key Findings

The reference use of BLIP demonstrates that a single pretrained model can support both captioning and simple VQA through minimal prompt design. A small amount of preprocessing and device aware configuration is enough to make BLIP usable in an interactive setting, even when hardware resources are limited and heavier solutions like Ollama cannot be deployed.

III. Experiment: BLIP Vision Explanation Studio

A. Experiment Objectives

The experiment aims to:

- build a single page web application where users upload an image and optionally ask a question,
- generate concise captions and answers using BLIP without additional fine tuning,
- present results in a professional dashboard style UI built with Gradio Blocks and custom CSS,
- make the active device and model visible in the UI header.

B. Experiment Tasks

The tasks encoded in the script are:

- load and configure BLIP for inference on CPU or GPU,
- implement the helper function `get_blip_caption`,
- implement the core function `analyze_image`,
- design the custom CSS theme for the app,
- build the two column layout with Gradio Blocks,
- connect the *Analyze Image* button to `analyze_image`.

C. Implementation Requirements

1) Conceptual BLIP Interface: Conceptually, BLIP chooses a text sequence that is most likely given an image and an optional question. This process can be summarized as the short pseudocode in Algorithm 1, which fits cleanly in a single IEEE column.

2) Main Application Pipeline (Detailed): The actual application logic is implemented by `analyze_image`, which handles input validation, optional resizing, captioning, question answering, and output formatting. The full pipeline is given in Algorithm 2.

3) UI Design: The UI is built with `gr.Blocks` and consists of:

- a header with app title, subtitle, and a status pill showing model and device,
- a left card with the image uploader, question textbox, analyze button, and short usage tips,

Algorithm 2 BLIP-based Image Analysis Pipeline

Require: image input `x` from UI, question string `q`
Ensure: output string `r` (caption, or caption + answer)

Input validation
if `x` is missing **then**
    return "Warning: please upload an image first."
end if
if `x` is not a PIL image **then**
    convert `x` to `PIL.Image` using a utility function
end if

Optional resizing
obtain width and height of `x` as `w` and `h`
if $\max(w, h) > 512$ **then**
    resize `x` so that the larger side is 512 pixels
end if

Caption generation
encode `x` with the BLIP processor into tensor inputs
move tensor inputs to the selected device
disable gradients using a no–gradient context
run BLIP text generation with a fixed beam size and token limit
decode the first generated sequence into caption text `c`

No–question case
if `q` is empty after stripping whitespace **then**
    format `r` as Markdown containing caption `c`
    return `r`
end if

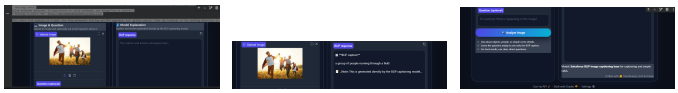
Question answering
build a prompt string of the form "Question: `q` Answer:"
encode image `x` and prompt with the BLIP processor
move the resulting tensors to the selected device
disable gradients using a no–gradient context
run BLIP text generation again with QA settings
decode the first generated sequence into answer text `a`

Output formatting
combine caption `c` and answer `a` into a Markdown string `r`
return `r`

- a right card with a heading, description, and a read–only textbox that shows the BLIP response,
- a dark theme with gradients, rounded cards, and styled buttons defined in a custom CSS string.

D. Analysis Report Requirements

1) Introduction and Theoretical Background: BLIP is a vision–language model that uses an image encoder and a text decoder to generate natural language descriptions of images. When only an image is provided, the model produces a caption. When both an image and a short question are provided, the same generation mechanism is used to produce an answer. Hugging Face Transformers exposes this capability through a single `generate` method, which the application calls in both caption only and caption plus question modes. This architecture offers a good trade–off between expressiveness and resource usage compared to heavier Ollama based LLM stacks.



(a) (b) (c)

Fig. 1. BLIP Vision Explanation Studio UI: (a) HTML output, (b) image upload, (c) simple VQA.

2) Experimental Setup: The experimental setup consists of:

- a Python environment with PyTorch, Transformers, Pillow, and Gradio,
- automatic device selection between CPU and GPU,
- the BLIP captioning model loaded in evaluation mode,
- a Gradio Blocks dashboard launched locally or via the `share` option.

Users upload images and optionally type questions; the system returns captions and answers in the output panel.

3) Implementation Details: The model logic is concentrated in two functions: `get_blip_caption` and `analyze_image`. The former wraps the BLIP processor and model to produce a caption; the latter implements the detailed pipeline in Algorithm 2. The UI is defined declaratively with Gradio Blocks, and the custom CSS controls the background, card appearance, typography, and button styling. The app header displays the active device so users can see whether GPU acceleration is in use.

4) Results and Analysis: Qualitative evaluation shows that BLIP produces short, coherent captions for many everyday images and often gives plausible answers to simple questions that are clearly grounded in the visible content. Resizing large images keeps the latency acceptable on limited hardware. The combination of clear headings, status indicators, and copyable output text makes the interface practical for demonstrations and small experiments.

5) Discussion and Conclusions: BLIP Vision Explanation Studio demonstrates that a single pretrained vision–language model can be wrapped in a small amount of code to deliver an interactive image captioning and simple VQA experience. The design works within tight resource constraints that rule out heavier Ollama based deployments, yet provides a useful and visually polished tool.

The same pattern can be reused with other models or extended with features such as multilingual captions, richer prompts, content filters, or additional evaluation capabilities.