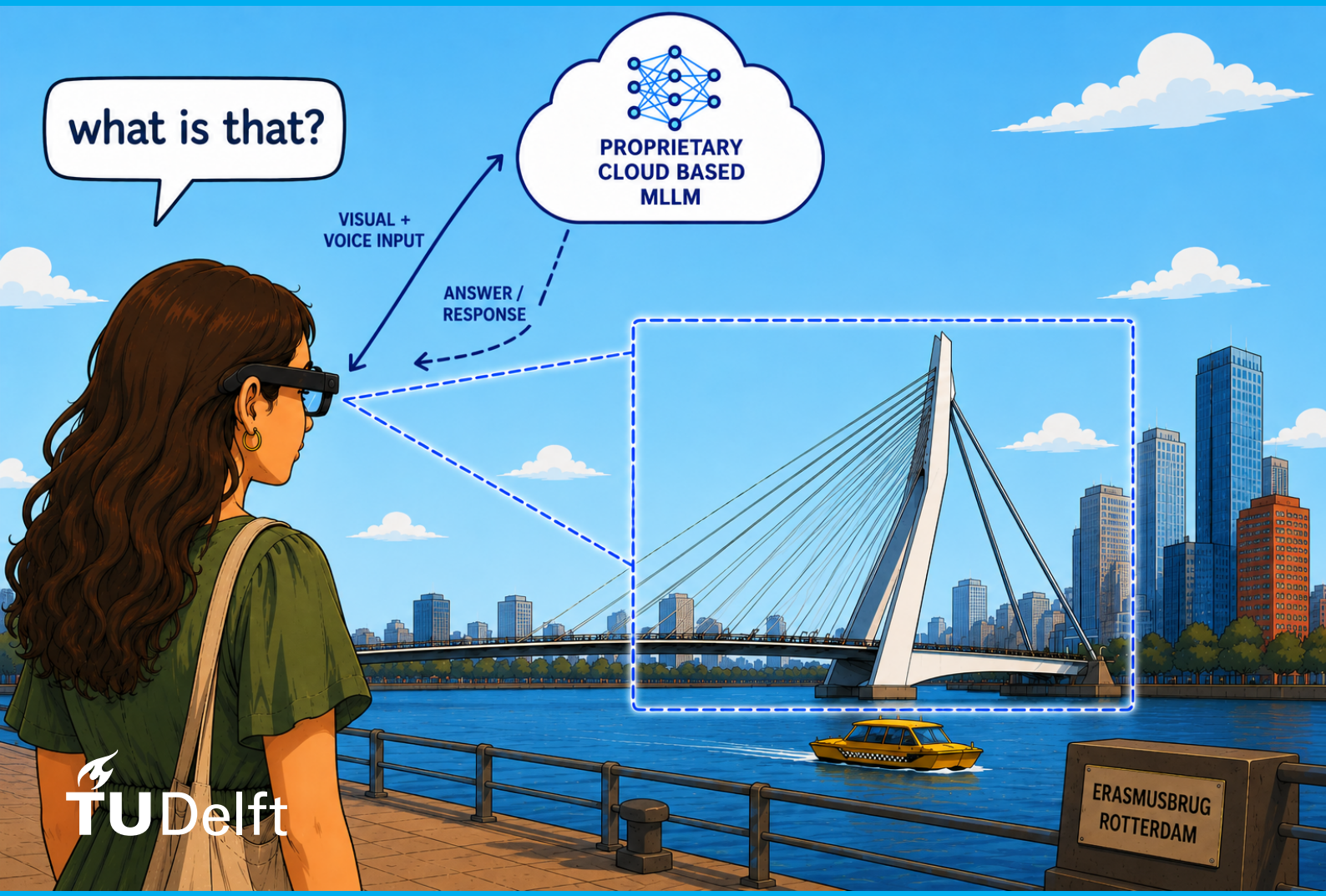
Visual Question Answering in Mobile AI Assistants: A Benchmark of Proprietary cloud-based multimodal LLMs

Evaluating Monetary Cost, Accuracy, Token Usage, Payload, and Latency

H. Vanhuynegem



Visual Question Answering in Mobile AI Assistants: A Benchmark of Proprietary cloud-based multimodal LLMs

Evaluating Monetary Cost, Accuracy, Token Usage, Payload, and Latency

by

H. Vanhuynegem

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday June 15, 2026 at 2:00 PM.

Student number:	5252687	
Project duration:	November 10, 2025 – June 15, 2026	
Thesis committee:	Dr. G. Lan,	TU Delft, supervisor
	Dr. M.A. Zuñiga Zamalloa,	TU Delft
	Dr. N. Mohan	TU Delft

This thesis is confidential and cannot be made public until June 15, 2026.

TU Delft

Preface

This thesis was conducted at the Department of Electrical Engineering, Mathematics and Computer Science (EEMCS) of Delft University of Technology, in partial fulfilment of the requirements for the degree of Master of Science in Computer and Embedded Systems Engineering. The research was carried out from November 2025 to June 2026.

First and foremost, I would like to thank Guohao Lan, my thesis supervisor, for his guidance and the time he dedicated to this project. I am glad that I considered the supervisor as a factor when choosing my thesis subject. His approach — encouraging me to take ownership of the direction of the research — pushed me to develop both as a researcher and as an individual. Through the many possible paths this project could have taken, I came to recognise one of my greatest personal challenges: making decisions in the face of open-ended possibilities. This experience has strengthened that weakness and left me more confident in navigating the unknown.

I would also like to thank Marco Zuñiga Zamalloa and Nitinder Mohan for serving on my Thesis Committee and for the time they invested in helping me reach this milestone.

I would also like to thank my family, friends and my girlfriend Julie for supporting me whenever I needed guidance!

Finally, I want to reflect briefly on the journey that brought me here. Completing both my Bachelor's in Electrical Engineering and this Master's degree over six years has been a long and at times demanding road. I am grateful for everything it has taught me, and I look forward to what comes next.

H. Vanhuynegem
Delft, June 2026

Abstract

Mobile AR assistants must offload visual queries to cloud multimodal large language models (MLLMs): on-device inference exceeds the power, memory, and thermal budgets of wearable hardware. This thesis measures how image preprocessing affects pipeline latency, payload size, token usage, cost, and visual question answering (VQA) accuracy when applied to the captured frame before transmission.

A controlled paired experiment—pairing each preprocessed sample with its unprocessed counterpart to eliminate confounding from per-image difficulty variance—compared 12 techniques across four provider-model configurations and three VQA datasets, logging all five dimensions per sample. The techniques span JPEG compression, downsampling, grayscale conversion, gaze-based region-of-interest (ROI) cropping, and saliency- and YOLO-based cropping. Relative to unprocessed images, JPEG at quality 85 reduces latency by 25% and payload by 50% with no detectable accuracy loss; gaze-based ROI cropping reduces latency by 38% and payload by over 85% at a 3-percentage-point accuracy cost, provided eye-tracking data are available. On Realtime-class streaming models, both techniques are recommended as deployment defaults.

This thesis introduces a principled taxonomy distinguishing *compression-only* preprocessing—which reduces payload without altering image geometry and therefore cannot discard task-relevant content—from *geometry-changing* preprocessing, which crops or resizes the image and can remove information the model would otherwise receive; this distinction predicts which technique classes incur accuracy costs and which do not. The open benchmark VQABench supports replication across additional providers, models, and strategies; the results are limited to still-frame VQA and should be validated separately for video or streaming queries. The findings extend beyond AR: any multimodal pipeline that transmits images to a cloud model can apply preprocessing-first optimisations before investing in prompt compression or model-level architectural changes.

Contents

1 Introduction

1.1. Background & Motivation

Mobile AI assistants, devices that help users interpret the physical world around them, must process visual scenes in real time. An assistant that cannot observe what the user sees cannot provide meaningful guidance during maintenance, training, navigation, or safety-critical work [1, 73]. Augmented reality (AR) assistants are a particularly demanding class within this broader category: head-mounted or handheld devices that overlay real-time guidance on a live camera feed. The scenes they capture are complex, and users ask questions such as "What is this component?" or "Is this connection correct?" Answering these requires vision-language reasoning that only multimodal large language models (MLLMs) can currently provide [116].

MLLMs cannot run on wearable hardware. Head-mounted AR devices are constrained by power budget, memory, and thermal limits that are incompatible with the inference requirements of state-of-the-art vision-language models [26]. Cloud offloading is the practical answer: prior work confirms that cloud-assisted AR can deliver the reasoning capability that local hardware cannot, and the combination is feasible for assistive applications [97, 113]. Every major commercial smart-glass product has reached the same conclusion: Ray-Ban Meta on Llama-class models [69], Rokid AI Glasses on ChatGPT and Gemini [91], RayNeo X2 on GPT-4 with Vision [105], Google's Android XR glasses on Gemini 2.5 Pro [38, 31]. The trade-off is that mobile AI assistance (AR assistance especially) becomes a distributed systems problem. Each query travels through local preprocessing, network transmission, cloud inference, and response delivery in sequence. Measurement studies report round-trip times of 250–500 ms to cloud services [94], slow enough that a faster model alone cannot compensate.

The developer building such a pipeline has two inputs to optimize: the textual query and the visual image. Textual input optimization is well-studied. Prompt compression, context-aware encoding, and query shortening have each been shown to reduce token usage and cost without degrading answer quality [46, 99, 20, 51]. For mobile AI assistants — and AR assistants especially — where voice is the natural interaction mode, this option is largely unavailable. The user's spoken query is transcribed and sent; the developer cannot shorten or restructure it without either requiring the user to speak differently or introducing a post-transcription editing step that adds its own latency. In practice, the textual query in a voice-driven mobile AI assistant pipeline is outside the developer's control.

Visual input preprocessing is therefore the primary design variable available to the developer. Before transmitting a captured frame to the cloud model, the application can compress it, downsample it, convert it to a lower-cost format, or crop it to the most likely task-relevant region [48, 108, 32, 107, 104]. These operations change the transmitted payload, the number of tokens billed by the provider, the total pipeline latency, and potentially the model's ability to answer correctly. The right choice depends on the joint effect across all these dimensions. Figure 1.1 shows this pipeline, with the preprocessing stage highlighted as the sole design variable.

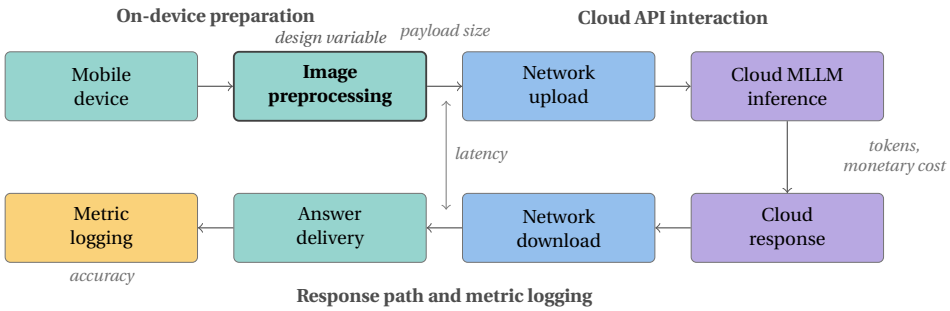


Figure 1.1: End-to-end pipeline for a single image-question query in a cloud-based mobile AI assistant. **Image preprocessing** (bold border) is the sole design variable available to the developer; the five annotated measurements — upload payload, token usage, monetary cost, end-to-end latency, and VQA accuracy — capture its joint effect on the pipeline.

1.2. Problem Statement

A developer choosing how to preprocess an image before a cloud MLLM call (whether for an AR assistant or another class of mobile AI assistant) has partial evidence on individual dimensions but no controlled benchmark measuring the joint effect. Several recent studies operate on commercial proprietary APIs, but each covers only a subset of the dimensions that matter for a deployment decision. Res-Bench [56] evaluates Visual Question Answering (VQA) accuracy under resolution variation but does not report payload size, token usage, cost, or latency. IPPg [21] measures token usage and monetary cost reductions but not upload payload or end-to-end latency. CodeOCR [98] measures token usage at different image compression ratios but not accuracy under general VQA conditions or total pipeline latency. Text or Pixels [57] reports token usage and partial latency but not the upload payload the device actually transmits. Zoomer [86] measures tokens and accuracy but leaves cost and latency unmeasured. OCR Resolution [43] compares accuracy at varying resolution without reporting any efficiency metric.

Taken together, the six studies above do not answer the question a developer actually needs to ask: "If I apply JPEG Q85 compression, what happens to my payload size, token count, monetary cost, total pipeline latency, and accuracy simultaneously?" The dimensions interact in non-obvious ways. A preprocessing method that reduces payload may not reduce tokens, because providers handle image formats differently. A token reduction may not reduce latency, because server-side processing time is opaque. A cost saving is worthless if the resulting accuracy falls below the threshold that makes the assistant useful.

1.3. Research Gap

Two bodies of prior work are relevant, and each falls short of answering the practical question for a different reason.

The first is visual token compression research, which is extensive. Surveys document over fifty distinct algorithms [95, 85], and methods such as FastV [16], LLaVA-PruneRGE [93], InternVL-X [63], and LFTR [121] demonstrate substantial token reductions. None of it transfers to the commercial API context. These methods work by modifying the visual encoder, the attention mechanism, or the token selection process inside the model — components that are inaccessible when using a hosted provider API. The efficiency gains are real, but they require access to model internals that commercial API users do not have.

The second body is recent work that operates on proprietary commercial APIs. These studies are the closest prior work, but each covers only a subset of the relevant dimensions (Table 1.1).

The payload column is empty across all six prior studies. Upload payload is what the device actually transmits, and its absence from every prior study means no prior work has connected the developer's action (choosing a preprocessing technique) to its first measurable consequence (the bytes sent over the network).

No existing study therefore provides a controlled comparison of standard image preprocessing techniques measuring payload, token usage, cost, latency, and accuracy simultaneously within the same commercial

Table 1.1: Dimension coverage of prior work operating on proprietary APIs. Upload payload has not been measured by any prior study.

Paper	Prop. API	Std. preproc.	Payload	Tokens	Cost	Latency	Accuracy
Res-Bench [56]	✓	✓	×	×	×	×	✓
Zoomer [86]	✓	Partial	×	✓	×	×	✓
IPPg [21]	✓	×	×	✓	✓	×	✓
Text or Pixels [57]	✓	×	×	✓	×	Partial	✓
CodeOCR [98]	✓	Partial	×	✓	✓	×	✓
OCR Resolution [43]	✓	✓	×	×	×	×	✓
This thesis	✓	✓	✓	✓	✓	✓	✓

cloud MLLM pipeline. This thesis addresses that directly.

1.4. Research Objectives & Questions

1.4.1. Research Objective

This thesis investigates image preprocessing as a systems-level design variable in cloud-based multimodal inference for mobile AI assistant-oriented applications. It provides the first controlled, paired comparison of preprocessing strategies measuring payload size, token usage, monetary cost, end-to-end latency, and VQA accuracy simultaneously within the same commercial cloud MLLM pipeline.

1.4.2. Research Questions

Three research questions guide the investigation:

1. **RQ1 – Efficiency impact:** To what extent do different image preprocessing techniques reduce payload size, token usage, monetary cost, and end-to-end latency in a cloud-based multimodal inference pipeline, and what local preprocessing overhead do they introduce?
2. **RQ2 – Accuracy impact:** To what extent do different image preprocessing techniques affect visual question answering accuracy relative to a no-preprocessing baseline?
3. **RQ3 – Overall suitability:** Which preprocessing techniques achieve the highest overall suitability across deployment scenarios, as determined by a ranked scoring approach that applies weighted scenario analysis to jointly score efficiency and accuracy across scenarios with different priorities — such as latency-critical, accuracy-critical, and cost-critical deployments?

1.5. Scope

This thesis contributes a controlled empirical measurement, not a deployed system or a new model. The experimental design treats the cloud provider as a black box and preprocessing as the independent variable. This matches the actual deployment context for mobile AI assistant developers and enables the paired comparisons that a live mobile AI assistant evaluation cannot support. A controlled study measuring all five dimensions simultaneously is what the prior literature lacks; a full production deployment would produce noisier, harder-to-interpret results. Proprietary commercial APIs are also preferred over self-hosted open-source models: hosted providers supply the operational reliability — stable uptime, managed scaling, and consistent inference environments — that real-world mobile AI assistant deployments depend on. Local model hosting introduces latency variability and operational overhead incompatible with both the controlled measurement goals and the deployment context the study is meant to reflect.

The study compares global image-reduction methods (downsampling, grayscale conversion, JPEG compression, WebP compression) and region-of-interest methods (saliency-guided, gaze-guided, and object-detection-guided cropping), measuring local preprocessing time, payload size, provider-reported token usage, monetary cost, end-to-end latency, and VQA accuracy. Comparisons use a paired design across the same image-question samples to ensure that observed differences reflect preprocessing choices rather than sample composition.

1.6. Contributions

The primary contribution is a controlled, paired comparison of standard image preprocessing techniques measuring the joint effect on upload payload, provider-reported token usage, monetary cost, end-to-end pipeline latency, and VQA accuracy within the same commercial cloud MLLM pipeline. No prior study has measured all five of these dimensions simultaneously in this setting.

The study also introduces a reusable measurement methodology: a paired black-box design that uses repeated same-sample runs across multiple providers to separate preprocessing effects from network and service-side variability. The protocol and logging pipeline are documented for reproducibility.

The empirical dataset, experimental protocol, and results are released publicly as **VQABench**[1] — a benchmark for evaluating image preprocessing strategies in commercial MLLM pipelines. VQABench is designed to be extensible: researchers and developers can reproduce the full measurement pipeline, incorporate additional preprocessing techniques, or extend coverage to other providers and model versions.

From the ranked scoring analysis — which applies weighted scenario analysis to jointly score efficiency and accuracy across scenarios with different deployment priorities — the thesis derives practical recommendations for mobile AI assistant developers: which techniques are safe defaults across scenarios, which dominate under latency-critical conditions, and where aggressive preprocessing hurts answer quality.

1.7. Thesis Structure

Chapter 2 defines three background concepts needed to interpret the experimental results: the preprocessing families used in the study, the five efficiency dimensions the thesis measures and why they do not vary together, and the task-performance constraint that determines whether an efficiency gain is actually useful.

Chapter 3 surveys the prior literature across mobile AI assistant offloading research, visual token compression, and recent commercial API studies. It confirms that the gap identified in Chapter 1 is real: no prior work provides the controlled, multi-dimensional comparison this thesis offers.

Chapter 4 covers the experimental methodology: the paired study design, hypotheses, preprocessing techniques, datasets, system setup, and statistical analysis, with each design choice justified against the gap it addresses.

Chapter 5 reports results for latency, payload, token usage, cost, and accuracy, connecting each finding to its hypothesis.

Chapter 6 interprets what those results mean, including where preprocessing methods fail, how the commercial API context changes expectations set by open-weight model research, and what system designers should take away.

The thesis closes in Chapter 7 with a summary of findings, direct answers to the three research questions, a statement of contributions, and directions for future work.

[1] Available at `https://github.com/Hvanhuynegem/VQABench` or `https://anonymous.4open.science/r/VQABench-8181`. A paper based on this thesis, *"How Much Does It Cost to Answer My Question? Benchmarking Preprocessing for Cloud VLM-based VQA"*, is submitted to the ACM Conference on Embedded Networked Sensor Systems (SenSys 2027) on 5 June 2026.

2

Background Information

This chapter establishes five concepts that the introduction does not cover in detail and that the reader needs to interpret the experimental results correctly. Section 2.1 defines what vision-language models are and explains why VQA is the evaluation task this thesis uses. Section 2.2 defines the preprocessing families used in the study and the distinct risk each introduces. Section 2.3 distinguishes the five efficiency dimensions the thesis measures and explains why they do not vary together. Section 2.4 explains the structural difference between REST and Realtime API interaction models and why the two produce different latency and token-accounting behavior. Section 2.5 defines task performance in the VQA setting, explains how accuracy is measured, establishes what constitutes acceptable degradation under efficiency-oriented preprocessing, and describes the limitations of benchmark-based evaluation.

2.1. Vision-Language Models and Visual Question Answering

Multimodal large language models (MLLMs) extend the text-based architecture of large language models to accept both image and text as input, producing text as output [116]. A visual encoder first processes the raw image into a sequence of patch-level representations, which are projected into the language model's token space; the model then reasons jointly over the visual and textual input to generate a response. The models evaluated in this thesis — `gpt-5.4-2026-03-05`, `gpt-realtime-mini`, `claude-sonnet-4-6`, and `gemini-3-flash-preview` — follow this general architecture, though their specific encoding and tokenization details are not publicly disclosed. This opacity is precisely why token usage is treated in this thesis as an empirical measurement rather than a quantity that can be derived analytically from image properties.

Visual Question Answering (VQA) is the interaction format used throughout this thesis to evaluate MLLM behaviour under varying image preprocessing conditions. Given an image and a natural-language question about it, the model must return a natural-language answer [8]. The format directly mirrors the usage pattern of mobile and AR assistants: a user observes a scene, poses a question, and expects an answer grounded in the visual content. VQA benchmarks operationalise this by pairing images with questions that have a verifiable ground-truth answer, making accuracy measurable in a controlled experiment.

VQA accuracy serves in this thesis as the constraint on acceptable preprocessing. Payload, token usage, cost, and latency can each be reduced by discarding visual information before sending an image to the model — but any reduction that causes the model to answer incorrectly is not a useful efficiency gain.

2.2. Image Preprocessing Techniques

Image preprocessing refers to input-side transformations applied before cloud inference: operations that reduce visual data, alter its encoding, or preserve selected regions likely relevant to the task [74, 45]. Two

families are relevant for this thesis.

Global reduction methods transform the entire image. JPEG and WebP reduce the encoded byte size while attempting to preserve perceptual quality [108, 32]. Downsampling reduces spatial resolution and therefore pixel count [107]. Grayscale conversion removes chromatic information [48]. These methods are simple and broadly supported, but their weakness is that relevant and irrelevant content are reduced together. Conventional resizing can degrade downstream recognition performance because downscaling removes high-frequency detail needed for visual understanding [104].

Selective preprocessing attempts to avoid this problem by preserving detail in a region of interest (ROI) while discarding or reducing less relevant content. Gaze-based methods use the user's visual attention as a relevance signal, which is particularly suited to AR-oriented assistance where the user's question often refers implicitly to what they are looking at [70, 114]. Saliency-based methods estimate visual prominence when no attention signal is available [41]. Task-independent saliency maps correlate poorly with the regions humans attend to when answering VQA questions, suggesting that a saliency crop may preserve visually prominent content while omitting the evidence the question actually requires [27]. Object-driven methods use a detector to identify semantically meaningful entities [3]. When a detector fails to localize objects necessary for reasoning, VQA accuracy drops, and the objects most reliably detected are not necessarily the most useful for answering the question [67]. Selective methods can preserve task-relevant detail while reducing overall visual data, but their value depends entirely on whether the selected region contains the evidence needed to answer the question.

Table 2.1 summarizes both families in terms of their expected efficiency mechanism and their primary validity risk.

Table 2.1: Preprocessing taxonomy used to interpret the experimental conditions. Each technique introduces a distinct validity risk that the experimental design must account for.

Family	Expected efficiency mechanism	Main risk
Encoding compression	Reduces transmitted bytes while preserving image dimensions	Compression artifacts may obscure text, edges, or small objects [108, 50]
Global resolution reduction	Reduces pixel count, encoded size, and potentially provider accounting	Fine detail is removed across both relevant and irrelevant regions [104]
Color reduction	Removes chromatic information and may reduce encoded complexity	Color-dependent questions or cues may become unanswerable [48, 58]
Attention-guided ROI	Preserves detail near the user's likely referent	Context outside the attended region may be omitted [119]
Saliency ROI	Keeps visually prominent regions when gaze is unavailable	Visually salient content may not match the question-relevant evidence [27]
Object-driven ROI	Keeps semantically detected entities while discarding background	Detector errors or missing scene context may remove needed evidence [67]

2.3. Pipeline Efficiency Dimensions

Efficiency in cloud-assisted multimodal inference cannot be captured by a single metric. A query in a mobile AI assistant pipeline passes through five sequential stages: local preprocessing on the client device, image encoding, network upload, cloud MLLM inference, and response delivery. Figure 1.1 shows the full pipeline. Each efficiency dimension this thesis measures corresponds to a specific stage or span of this pipeline, and a change at one stage does not propagate predictably to the others. This thesis measures five dimensions: upload payload, provider-reported token usage, monetary cost, local preprocessing overhead, and end-to-end latency.

Upload payload is the byte count transmitted from the client to the cloud provider at the network upload stage. It is determined by the image encoding applied before transmission. Payload measures the transport cost of a preprocessing decision: it governs upload duration and determines the request's sensitivity to network variability, which is consequential in mobile and AR deployments where connectivity may be inter-

mittent or bandwidth-limited [97, 90, 117].

Provider-reported token usage is recorded at the cloud MLLM inference stage and reflects the billing units the provider assigns to a request after applying its own tokenization scheme. For text, all three providers use a subword tokenization algorithm in which one token corresponds roughly to four English characters; OpenAI reports this approximation explicitly [84], and Gemini states that 100 tokens correspond to 60–80 English words [37]. For images, each provider applies a distinct resolution-dependent scheme, and the scheme can differ across models within the same provider. Larger OpenAI models such as `gpt-5.4-2026-03-05` scale the image to fit within a 2048×2048 pixel bounding box, reduce the shorter side to 768 pixels, and count the resulting 512×512 pixel tiles, with each tile adding a fixed token charge on top of a per-image base cost [77]. OpenAI's smaller mini and nano models, including `gpt-realtime-mini`, instead cover the image with non-overlapping 32×32 pixel patches under a fixed patch budget and scale the resulting count by a model-specific multiplier [77]. Anthropic tokenizes approximately as $\lceil w \cdot h/750 \rceil$ tokens, where w and h are the dimensions of the decoded pixel grid [7]. Gemini charges a flat 258 tokens for any image whose longer side is at most 384 pixels; for larger images, it divides the image into 768×768 pixel tiles, each counted at 258 tokens [36]. Because each scheme tiles or buckets the decoded pixel grid in discrete steps, the same pixel content can map to substantially different token counts across providers, and shrinking an image below a tile or bucket boundary reduces its token count discontinuously. subsection 5.3.1 and section A.5 verify these schemes empirically: `gpt-5.4-2026-03-05` and `claude-sonnet-4-6` show the stepwise token increases expected from tiled accounting, whereas `gpt-realtime-mini` and `gemini-3-flash-preview` report near-constant counts across the tested resolution range.

In all cases, the provider tokenizes the image as received after transmission rather than the compressed payload bytes, so a smaller compressed payload does not guarantee a proportionally smaller token count. Provider-reported token usage therefore cannot be predicted from payload alone and must be treated as an empirical measurement rather than a quantity derivable from image properties. Token usage is also the direct precursor to monetary cost.

Monetary cost is derived from provider-reported token usage by applying provider-specific per-unit pricing [79, 5, 35]. Pricing structures vary by provider and may include tiered rates or per-call charges, so a token reduction does not guarantee a proportional cost reduction. Cost scales with query volume: in continuously operating mobile AI assistants, per-query monetary cost accumulates into a binding economic constraint even when individual amounts are small.

Local preprocessing overhead is the computation time added at the on-device preparation stage before upload. Selective methods that require saliency estimation, gaze-guided cropping, or object detection introduce measurable client-side computation; benchmarks on edge hardware show inference latency spanning hundreds of milliseconds to several seconds depending on model size and device class [3, 24]. On battery- and thermally-constrained devices such as AR assistants, this overhead competes with other foreground processes and directly affects whether a preprocessing technique is practical [92, 1].

End-to-end latency spans from the moment the encoded image and prompt are transmitted to the cloud provider until the full response has been received. It does not include local preprocessing overhead; that component is tracked separately as its own efficiency dimension. End-to-end latency therefore decomposes into network upload, cloud MLLM inference, and response delivery [94, 90, 97]. Measurement studies report cloud round-trip times of 250–500 ms under normal conditions [94]. Preprocessing can reduce end-to-end latency by shortening upload time or reducing the volume of tokens the model must process.

The relative importance of the five dimensions depends on the deployment context. Table 2.2 maps each representative deployment scenario to its binding efficiency dimension and notes where task accuracy imposes an additional constraint that preprocessing must respect.

Table 2.2: Relative importance of efficiency dimensions by deployment context. Rankings indicate priority within each application: 1 = most important, 3 = least important. No deployment context prioritises all dimensions equally; preprocessing choices must be matched to the constraints of the target application.

Platform	Mobile Phone AI assistant			Wearable AR assistant	
Application	Real-time AR overlay[a]	Accessibility assistance[b]	Document / text scanning[c]	Navigation & object detection[d]	Task guidance[e]
Accuracy[f]	3	1	1	2	1
Latency[g]	1	2	3	1	2
Cost[h]	2	3	2	3	3

[a] Visual guidance requires near-synchronous updates; payload reduction shortens upload time [94, 90].
[b] Low latency and hallucination-resistant accuracy are jointly required [120].
[c] Latency-tolerant; fine-grained textual detail must survive preprocessing [43].
[d] Delayed guidance in dynamic environments constitutes an incorrect response [94, 90].
[e] A fast incorrect instruction is more harmful than a correct delayed one [113].
[f] VQA soft accuracy relative to the unprocessed baseline; a cross-cutting constraint that all preprocessing must respect, not one of the five efficiency dimensions.
[g] Combines local preprocessing overhead (dimension 4) and end-to-end latency (dimension 5), both of which determine user-perceived response time. Upload payload (dimension 1) is a determinant of end-to-end latency and is subsumed here.
[h] Monetary cost (dimension 3) derived from provider-reported token usage (dimension 2); token usage is the direct precursor to cost, so both dimensions are represented by this row.

2.4. REST and Realtime API Interaction Models

The cloud-based inference pipeline in this thesis uses two structurally different API interaction models. The choice between them affects how latency decomposes, how tokens are counted, and whether cached context from prior calls can reduce the cost of subsequent ones. Understanding the difference is necessary to interpret why the two API types produce different efficiency profiles in the results.

A **REST API** follows a stateless request-response pattern over HTTP [75]. Each call is independent: the client opens a connection, transmits the full request payload including the image and prompt, waits while the provider processes it, and receives the streamed or complete response before the connection closes. Because there is no persistent state between calls, every request pays its own connection overhead and no image or prompt content carries over from one call to the next. Maintaining conversational context therefore requires the client to include the full message history in every subsequent request; as the conversation grows, so does the input payload of each new call, increasing both upload cost and token usage proportionally. From the client's perspective, the externally observable latency spans from the start of payload transmission to the end of response receipt. It decomposes into client send time, server-side processing up to the first response token, and streamed response delivery.

A **Realtime API** uses a persistent WebSocket connection [81, 80]. Rather than sending discrete HTTP requests, the client establishes a session once and then exchanges structured JSON events over the same open channel. Session parameters are configured by sending `session.update` events, which specify what context the provider should retain server-side for the duration of the session. The connection setup cost can therefore be amortized across multiple calls within the session rather than paid per request. Within a session, the per-request latency decomposition is more detailed than for REST: it includes request event creation, server acknowledgment of the event, the interval from the last event to the first response token, and streamed delivery. Because the connection persists, stable session context such as system instructions can be cached server-side across calls. The provider may report a portion of input tokens as cached, which are typically billed at a discounted rate. This cached-token behavior does not occur in REST calls, where each request is processed independently.

These structural differences have two practical consequences for this thesis. First, the latency fields recorded for the Realtime model include stages that do not exist for REST models, so the two API types require separate latency decompositions. Second, the Realtime model can exhibit non-zero cached token counts even when no chat history is intentionally maintained, because stable session instructions may be recognized as

reusable context by the provider. The token and cost accounting for the Realtime model must therefore handle cached input separately, whereas REST models treat all input tokens as uncached. In this thesis, the REST API is used for `gpt-5.4-2026-03-05`, `claude-sonnet-4-6`, and `gemini-3-flash-preview`; the Realtime API is used for `gpt-realtime-mini`.

2.5. Task Performance and Acceptable Degradation

Task performance in VQA refers to a model's ability to produce correct answers to visual questions. It is measured as accuracy over a benchmark set of image-question pairs, each with a verifiable ground-truth answer [39]. The baseline throughout this thesis is the accuracy the model achieves on unprocessed images; every preprocessing condition is evaluated by comparing its accuracy against that baseline on the same samples. This paired design ensures that observed accuracy differences reflect preprocessing choices rather than variation in question difficulty across runs.

VQA benchmarks have known limitations as measurement instruments. Ground-truth answers are crowdsourced from multiple annotators, so some answer variability reflects genuine inter-annotator disagreement rather than model error [8]; VQA soft scoring, described below, awards partial credit specifically to accommodate this. Benchmark datasets can also exhibit distributional bias in question types or image content [39], and performance on the benchmark may not generalize to a different image corpus [62]. These limitations do not invalidate accuracy as a task-performance measure, but they matter when interpreting small absolute differences: a gap of one or two percentage points may lie within the noise introduced by annotation variability and should not automatically be treated as meaningful degradation.

The evaluation question is therefore not only whether a method reduces payload, tokens, cost, or latency, but whether it does so without causing an unacceptable drop in answer quality relative to the unprocessed baseline. Following the convention in the visual token compression literature, where drops below 1 percentage point are routinely described as not sacrificing performance [16], this thesis treats 1 percentage point as the upper bound of acceptable degradation when accuracy is the primary deployment constraint. In contexts where latency or cost rank above accuracy — such as real-time AR overlays, where Table 2.2 ranks accuracy third — a larger accuracy drop may be tolerable if the efficiency gain is proportionally large. A preprocessing result is Pareto-acceptable if it improves at least one efficiency dimension without exceeding the threshold appropriate for its deployment context. Where multiple methods are Pareto-acceptable, the one that achieves the largest combined efficiency gain is preferred.

For VQA benchmarks with short, verifiable answers, task performance is quantified using **VQA-style soft accuracy** [8]. Each model answer is normalized and compared against ten human reference answers collected for the same image-question pair. If m denotes the number of human answers that match the model prediction after normalization, the per-sample score is:

$$\text{accuracy} = \min\left(\frac{m}{3}, 1\right). \tag{2.1}$$

The formula yields per-sample scores of 0, 0.33, 0.67, or 1.0, averaged over all samples in a condition to produce the reported accuracy for each preprocessing technique. The formula rewards agreement with the majority of annotators without requiring unanimous consensus, which makes it robust to the natural variation in how different people describe the same visual content. It is preferred over simple exact-match accuracy specifically because of this tolerance for inter-annotator ambiguity.

For open-ended or scene-description tasks, where there is no single correct short answer, VQA soft scoring is not applicable. Rubric-based LLM judging and pairwise comparison are used instead [17]. `gpt-4.1-2025-04-14` serves as the judge and evaluates the candidate answer against the reference answer on four criteria — coverage, accuracy, details, and fluency — which are combined into a weighted total score; Table 2.3 shows the criteria, their definitions, and the weighting formula. Pairwise comparisons against a no-preprocessing baseline are evaluated in both answer orderings to mitigate sequence-order bias, and the results are aggregated into win, tie, or loss outcomes. The specific rubric prompts and scoring formulas used for each dataset in this thesis are defined in the evaluation-metrics section of the experimental methodology.

Table 2.3: `gpt-4.1-2025-04-14` judge rubric for open-ended evaluation [17]. The total score is a weighted sum of the four criterion scores. Coverage and accuracy together account for 80 % of the score, reflecting that factual correctness is the primary evaluation criterion.

Criterion	**Weight**	**Scale**	**Description**
Coverage	40%	0–10	Whether the response addresses all relevant aspects of the question.
Accuracy	40%	0–10	Correctness relative to the ground-truth answer.
Details	15%	0–10	Specificity and richness of information provided.
Fluency	5%	0–10	Language quality and readability.
Total score	100%	0–10	$0.40 \cdot \text{coverage} + 0.40 \cdot \text{accuracy} + 0.15 \cdot \text{details} + 0.05 \cdot \text{fluency}$
Pairwise preference	—	W/T/L	The judge model compares the candidate against the baseline in both orderings; outcomes are aggregated as win, tie, or loss.

With these five concepts established — the MLLM architecture and VQA evaluation format, the two preprocessing families, the five efficiency dimensions and their deployment-specific priorities, the structural difference between REST and Realtime API models, and the acceptable-degradation threshold for task performance — Chapter 3 situates this thesis within the prior literature.

3

Related Work

The convergence of vision-language models and mobile wearable devices has produced a rapidly growing research area comprising systems that observe the user's scene, capture their attention and voice commands, interpret natural-language queries about it, and respond contextually [96, 100, 115]. This chapter reviews the literature directly relevant to this thesis, organizing it by the role preprocessing plays in each system and the deployment surface the system targets. Five threads are particularly relevant to the present thesis: LLM-assisted task guidance and procedural assistance; context-aware and accessibility-focused visual assistants; on-device vs. cloud-offloaded deployment architectures; data preprocessing techniques for managing the cost and quality of cloud VLM inference; and VLM benchmarks that evaluate capability, efficiency, and cost.

3.1. LLM-Based Mobile AI Assistants

LLM-assisted task guidance and procedural assistance. A substantial line of work uses LLMs and VLMs to provide context-aware guidance for physical tasks observed through a head-worn camera. Egocentric video, speech, and LLM-driven contextual analysis can be fused to estimate task state and deliver adaptive AR guidance [72]. Multi-step AR instructions can be generated by combining an LLM with vision models to identify visual augmentation types, extract spatial information about key interaction points, and embed visuals in the physical space to support task execution [55]. Gaze-driven saliency can serve as an implicit attention signal for context-aware LLM agents in multi-window XR workspaces [13]. Existing works have also explored how to scale training data for such assistants: synthetic dialogue generation from instructional videos [2, 111] produces large procedural-assistance corpora without the cost of human-human data collection. This thesis does not contribute to task-guidance methods; it characterizes the cost and quality implications of the image preprocessing step that precedes any VLM query in these systems.

Context-aware visual assistants. Beyond standard procedural guidance, several works use gaze, gesture, and other implicit context signals to disambiguate user queries to a visual assistant. Real-time eye gaze, pointing-gesture recognition, and conversation history can be combined to resolve pronoun references in spoken queries: when the user asks "what is this?," computer vision over the user's field of view replaces "this" with a concrete referent before the rewritten query reaches an LLM [54]. This operates at the query level (text rewriting) rather than the image level (visual cropping), which is where this thesis intervenes. Context awareness can be extended to spatiotemporal personalized assistance by maintaining representations of past observed scenes for later retrieval [14]. Cross-format RAG systems integrate XR sensor streams with external knowledge bases for maintenance assistance [71]. Across these systems, the recurring theme is that the user's implicit attention—e.g., gaze, gesture, conversational history, recent activity—provides signal that a pure image-plus-text query lacks, and that incorporating this signal improves both the relevance and the efficiency of cloud VLM queries. This thesis evaluates gaze as one such signal at the image level, measuring its cost-quality trade-off on commercial cloud VLMs.

Accessibility-focused visual assistants. A particularly impactful application of VLM-based assistants is support for blind and low-vision (BLV) users. GPT-4-class VLMs have been commercially deployed to provide on-demand visual interpretation through a smartphone or smart-glass camera [10]. Glass-based visual assistance for blind users is also under active development, combining universal-assistant research with specialized accessibility services [38]. The spatial-reasoning capabilities of GPT-4o, GPT-4V, Claude, and Gemini on high-stakes BLV navigation tasks reveal systematic spatial biases—such as a preference for "left" over "right" descriptions—and over-reliance on color-based cues when stronger spatial descriptors are needed [68]. Specific failure modes arise when BLV-captured images are blurry, poorly framed, or ambiguous—failure modes that interact directly with preprocessing choices—as documented in long-form and consistency-aware VQA studies [42, 19]. These works illustrate that the practical usefulness of a VLM assistant depends not only on model accuracy in the abstract, but on how the visual input reaches the model; the preprocessing pipeline that determines how that input arrives is precisely what this thesis benchmarks.

3.2. Deployment Choices for AI Assistants

A central architectural question for mobile AI assistants is whether inference should run locally or in the cloud: local inference offers better privacy, offline operation capability, and lower network-transit latency, whereas cloud inference grants access to substantially more capable models. For local deployment, mobile-tailored VLMs at 1.4B/2.7B scale with an efficient CLIP-based vision encoder have been demonstrated on smartphone-class hardware [23]; 3.1B-scale models can match or exceed standard 7B baselines on common VLM benchmarks [122]; and sub-billion-parameter language models optimized for on-device deployment can achieve accuracy comparable to LLaMA-2-7B on certain API-calling tasks while running at 50 tokens/s on iPhone-class hardware [61]. This thesis focuses exclusively on cloud-offloaded deployment, where the most capable current models reside and where input preprocessing directly controls the cost of each API call.

Despite this progress, on-device deployment is not yet the dominant option for capable mobile AI assistants. A recent comprehensive study of more than 60 small language models for edge deployment [65] finds that, while state-of-the-art SLMs can outperform 7B-class models on general tasks, their in-context learning capabilities remain limited and their efficiency still has significant optimization potential — gaps that translate directly into worse multi-step reasoning and tighter latency margins for interactive use. The most capable VLMs — GPT-5, Claude Sonnet, Gemini Flash — remain accessible only through cloud APIs, and even high-end mobile SoCs cannot match their reasoning quality under interactive latency budgets. The result is evidenced by design choices in the commercial market: every major smart-glass platform offloads multimodal reasoning to cloud backends. Ray-Ban Meta runs on Meta's Llama-class models [69]; Rokid AI Glasses use ChatGPT and Gemini [91]; RayNeo X2 uses GPT-4 with Vision [105]; Google's Android XR glasses run on Gemini 2.5 Pro with Project Astra [38, 31]; and Apple's announced 2027 smart glasses tether to the iPhone, which itself escalates complex visual queries to Private Cloud Compute or ChatGPT [30, 106]. This thesis therefore studies image preprocessing exclusively in the cloud-offloaded setting that these products adopt, where every query incurs an API cost that input-side preprocessing can directly reduce.

3.3. Data Preprocessing for Cost Saving

Preprocessing the image input before offloading is a natural design choice for managing the end-to-end system cost: a single preprocessing method can significantly reduce payload size, token usage, server-side processing time, but also the visual evidence available to the model. Existing techniques fall into two families that differ in how they treat task-relevance.

Global reduction methods discard information uniformly across the image. Lossy compression formats such as JPEG [108] and WebP [32] shrink the byte footprint at a given perceptual quality and are natively accepted as image inputs by all three major cloud VLM providers [77, 7, 36], making them the most deployment-ready payload reduction. Spatial downsampling shrinks the pixel grid itself, reducing both payload and image-token count [107]; grayscale conversion removes the chromatic channels, shrinking the representation by roughly 3× [48]. These methods are simple and effective at reducing input size, but they also risk removing essential content. Aggressive downscaling significantly degrades recognition by stripping the high-frequency

detail that fine-grained reasoning depends on [104], and reducing or removing chromatic information measurably lowers accuracy on color-dependent multimodal tasks [58]. Global reduction therefore offers a calibrated but blunt trade-off between bytes saved and content preserved.

Region-selective methods take the opposite approach: they localize a task-relevant region of interest (ROI) and transmit only that region, optionally accompanied by a coarse global thumbnail for context. Because the transmitted image carries fewer pixels, region-selective approaches can in principle achieve simultaneous byte *and* token reductions. Two image-derived signals are commonly used to define the ROI without requiring any additional sensor hardware. *Saliency-based* selection uses spectral-residual or learned saliency maps to estimate visual prominence without any task context [41]. *Object-detection-based* selection runs a lightweight detector (e.g., a YOLO-family model) and extracts crops around detected entities [3, 45]; however, DNN-based object detection adds local CPU/GPU inference overhead that is hardware-dependent and may dominate the savings on constrained devices. Both signals are image-derived and require no additional sensor hardware, but they provide no task context and may select regions irrelevant to the user's question.

Gaze-guided selection uses the user's own eye fixation, captured by an eye tracker, to define the ROI [103, 18]; because gaze reflects where the user is actually attending, it serves as a direct semantic signal of relevance rather than an image-only estimate. A gaze-guided two-scale representation—a high-resolution ROI crop combined with a low-resolution thumbnail—reduces visual tokens by up to 93% on open-weight VLMs while maintaining task accuracy [18]. Aligning model attention with human gaze has also been shown to improve grounding and reduce hallucination in open-ended visual assistance [114]. This thesis evaluates gaze-guided cropping on commercial cloud VLMs, extending this evidence beyond the open-weight setting in which these results were established.

These results establish ROI selection as a promising lever, but the evidence comes entirely from open-weight VLMs with linear pixel-proportional token accounting and full internal access — a setting that does not match commercial VLM APIs, where tokenization is step-function (tile-based grids or whole-image units) and model internals are opaque [77, 7, 36]. Region-selective methods also carry task-dependent risks: gaze cropping may discard content needed for global questions such as counting or spatial comparison, on which VLMs already struggle even with the full image present [22]; object detection post-processing such as non-maximum suppression cannot fully eliminate overlapping or redundant boxes in crowded scenes, so returning multiple object crops may yield a combined payload that exceeds the original image [3, 29]; and bottom-up saliency selects visually prominent regions that need not correspond to the regions semantically relevant to the question [29, 112]. Whether any region-selective technique delivers net cost–quality gains on commercial cloud VLMs — and how the gain depends jointly on the selection signal, the task, and the visual content — remains an open question that this thesis answers empirically.

3.4. VLM Benchmarks

A growing body of work benchmarks VLMs along different axes — task accuracy, robustness in mobile scenarios, and on-device inference efficiency. Table 3.1 compares this thesis against four categories in which these efforts cluster. A common gap runs through all existing works: every prior benchmark varies either the model or the hardware while holding the input fixed, and none reports the joint cost-quality outcome on the commercial cloud APIs that current LLM-based AI assistants actually run on.

Classical and capability VQA benchmarks measure what a VLM can answer, not what an answer costs. VQAv2 [39], OK-VQA [66], VizWiz [40], and EgoVQA [28] report soft accuracy on open-ended questions with model-internal preprocessing only. Their capability-focused successors — MMBench [60], MMMU [118], and MathVista [64] — extend the evaluation to fine-grained skill, discipline, and reasoning taxonomies, but still report accuracy as the single output dimension. Latency, payload, tokens, and monetary cost are not measured; the input pipeline is fixed.

Wearable and smart-glass VQA benchmarks adopt the egocentric setting but inherit the same single-dimensional focus. HoloAssist [110] measures task-completion on AR captures; WearVQA [15] reports accuracy across six naturally occurring quality issues (unzoomed, occluded, rotated, cut-off, blurred, low-light); SUPERGLASSES [47] evaluates a large set of VLMs on smart-glass imagery and proposes an agent that

Table 3.1: Comparison of this thesis against related benchmarks. This thesis is the only work that varies preprocessing as an experimental variable while measuring its effect across five cost dimensions on commercial cloud VLMs.

Benchmark	Data scope	Benchmark size (QA pairs)	What is benchmarked	Image preprocessing	API
Classical VQA datasets					
VQAv2 [39]	Third-person	1.1M	Accuracy on open-ended VQA	Resize + normalize	Local
OK-VQA [66]	Third-person	14,055	Accuracy on knowledge-based VQA	Resize + normalize	Local
VizWiz [40]	Egocentric	31,173	Accuracy, answerability on BLV-captured VQA	Resize + normalize	Local
EgoVQA [28]	Egocentric	600	Accuracy on egocentric video QA	Standard + frame sampling	Local
Capability-focused VLM benchmarks					
MMBench [60]	Third-person	2,974	Accuracy (20 skills) on vision-language tasks	Resize + normalize	REST + Local
MMMU [118]	Third-person	11,500	Accuracy (6 disciplines) on college-level multi-discipline tasks	Resize + normalize	REST + Local
MathVista [64]	Third-person	6,141	Accuracy (7 reasoning types) on mathematical visual reasoning	Resize + normalize	REST + Local
Wearable, AR, and smart-glass VQA benchmarks					
HoloAssist [110]	Egocentric	166 hrs video	Task-completion, mistake detection on AR task guidance	Resize + frame sampling	Local
WearVQA [15]	Egocentric	2,500	Accuracy on wearable images across six quality issues	Resize; six natural degradations	REST + Local
SUPERGLASSES [47]	Egocentric	2,422	Accuracy, retrieval trajectory on VQA	Object detection	REST + Local
CRAG-MM [109]	Egocentric, Third-person	6,500 + 2K conv.	Truthfulness, hallucination rate on multimodal RAG	Resize; natural quality variation	REST + Local
ML inference benchmarks					
MLPerf Inf. [87]	Third-person	–	Throughput, latency, accuracy on standardized inference	Standard ImageNet pipeline	Local
DAWNBench [25]	Third-person	–	Time-to-accuracy, latency, cost on image classification	Standard pipeline	Local
nnPerf [24]	Third-person	–	Per-operator latency, energy on on-device DNN inference	Profiled at operator level	Local
MELT [52]	–	–	Throughput, latency, energy, memory on on-device LLM	N/A (text-only)	Local
This thesis	**Egocentric, Third-person, Attention**	**4,514**	**Accuracy, latency, payload, tokens, $ under preproc./API/provider variation on cloud VL**	**12 techniques: com downsampling, ROI**	**REST + Realtime**

incorporates object detection as a fixed pipeline component; and CRAGMM [109] measures multimodal RAG factuality over multi-turn dialogues. All four are valuable contributions to the wearable VQA setting, but each treats input as captured: preprocessing is not a design variable, and cost is not reported.

ML inference benchmarks measure efficiency, but their cost model does not transfer to commercial cloud VLMs. MLPerf [87], DAWNBench [25], nnPerf [24], and MELT [52] report throughput, latency, energy, and memory across hardware and on-device models. These quantities are determined by operator FLOPs and memory bandwidth on the local device. Cost on a commercial VLM API is determined by an entirely different mechanism — provider-specific tokenization (tile-based grids for OpenAI, whole-image accounting for Anthropic) plus per-token billing plus network round-trip latency. Findings from on-device efficiency benchmarks therefore do not generalise to the cloud-offloaded regime in which production VLM applications now operate.

3.5. Implications and Research Gap

We have three implications that directly motivate the present thesis. First, cloud offloading to commercial VLM APIs is the dominant deployment choice to enable capable multimodal AI assistants — across the entire current generation of commercial smart-glass products — yet those APIs expose pricing, tokenization, and latency behavior that are opaque to the practitioner. Second, the most promising context-aware assistant designs, e.g., gaze and attention-based ROI information extraction, manipulate the original visual input *before* sending it to the LLM model, making input-side preprocessing the principal and practical design choice available to practitioners building on top of these APIs. Third, unfortunately, no prior benchmark measures

how that lever maps to cost on commercial cloud VLMs. Existing benchmarks focus on LLM model accuracy on a fixed input pipeline, and the preprocessing literature focuses on open-source VLMs, whose linear pixel-proportional token accounting does not generalize to commercial providers with opaque tile-based or whole-image tokenization.

This thesis fills this gap. It is the first benchmark to treat data preprocessing as the experimental variable while measuring its effect across five cost dimensions (accuracy, latency, payload, tokens, and monetary cost) on commercial cloud VLMs. It evaluates 12 preprocessing techniques on two API paradigms (REST and Realtime) across three providers (OpenAI, Anthropic, Google) under a paired benchmark design that isolates the contribution of each system factor. By doing so, we establish the empirical foundation needed to make principled preprocessing decisions in cloud-offloaded multimodal inference systems.

4

Experimental Methodology

This chapter describes the measurement study used to evaluate image preprocessing techniques for cloud-based multimodal inference in AR-oriented scenarios. Preprocessing is treated as an input-side systems optimization problem: each technique is applied before the request is dispatched, and its effect is measured across payload size, token usage, cost, latency, and answer accuracy. The remote model is a black box throughout, so all results rest on externally observable client-side measurements.

The chapter is organized in six sections. Section 4.1 establishes the controlled paired comparison design and its connection to the research questions. Section 4.2 presents the three datasets and the criteria that determined their selection and roles. Section 4.3 defines the models selected, the independent and dependent variables, and the evaluation metrics. Section 4.4 narrates the procedure from instrumentation and data preparation through trial execution and controls. Section 4.5 specifies how results are expressed, compared, and summarized. Section 4.6 closes with an assessment of threats to validity.

4.1. Study Framing

This thesis follows a controlled experimental measurement-study design to analyse 12 image preprocessing techniques across three VQA datasets, evaluating their effect on multimodal inference pipeline efficiency using commercial vision-language model APIs. The remote model is treated as a black box, and preprocessing is studied as an input-side systems optimization problem. The study focuses on measurable effects on local preprocessing overhead, payload size, token usage, monetary cost, latency, and VQA answer accuracy.

The methodology is comparative and paired: each preprocessing technique is applied to the same image-question samples. This reduces between-sample noise and makes observed differences more directly attributable to preprocessing rather than dataset composition. The paired design is important because both VQA answer accuracy and latency can vary substantially across individual samples. This design directly addresses the gap identified in Chapters 1 and 3: no prior study provides a controlled, multi-dimensional comparison of preprocessing techniques within a single commercial cloud pipeline.

Experiments are conducted under a fixed setup, using consistent hardware, software, model configuration, prompt structure, and evaluation protocol within each experiment batch. Cloud-based evaluation cannot remove all external variability — especially network conditions and service-side behavior — so the methodology combines controlled design choices with repeated measurements, variance-aware analysis, and explicit reporting of uncertainty.

(a) VQA-MHUG

Short answers · 10 human references

Q: What is the design on the mug?

A: frown face; sad face; frowny face … (+7)

(b) VOILA-A

Long open-ended · 1 annotated answer

Q: What is the background of the view?

A: A wooded area with trees behind the two posing people. (full sentence)

(c) DriVQA

Short answers · 66–68 human references

Q: Tell me the name of the store on the right?

A: wagamama; not really; wagamama; wagamama … (+63)

Figure 4.1: Representative image–question–answer triples from the three datasets, illustrating their differing *answer styles*. VQA-MHUG provides ten short human reference answers per pair and DriVQA 66–68, while VOILA-A (VOILA-COCO version) provides a single long, open-ended annotated answer. The colored tag under each name summaries the style; `(+N)` denotes additional references not shown.

4.2. Datasets

The datasets are used as reproducible proxies for multimodal interaction under different preprocessing conditions. Because no single public dataset fully captures all aspects of gaze-enabled AR interaction, the study uses three datasets with different roles. **VQA-MHUG** is the primary dataset for the quantitative comparison of image preprocessing techniques. **DriVQA** covers a domain-specific driving setting too small for general claims, serving as a concrete external check. **VOILA-A** extends the evaluation toward open-ended gaze-conditioned assistance, where longer natural-language responses are evaluated. The three datasets are not compared against one another; each is used within the task setting for which it was designed. Figure 4.1 shows a representative sample from each dataset, illustrating the differences in image domain, question style, and expected answer format.

4.2.1. Selection Criteria

Each dataset must contain gaze data, be in English, be publicly available, and be relevant to the AR-oriented context of the thesis. Because few datasets meet all these criteria, the selection strategy is deliberately hierarchical.

Given this hierarchical structure, the primary dataset must produce meaningful results on its own. It must contain real human gaze data collected during visual question answering, with human reference answers available for evaluation. The evaluation technique should operate without LLM-based judging, to allow a controlled paired comparison across preprocessing methods using a standard answer-accuracy metric. Each preprocessing method must be applicable to the dataset. The second and third datasets must contain gaze data to be relevant to the motivation of the thesis, but they can be used for more open-ended evaluation with LLM-based judging rather than VQA-style accuracy. The datasets must be publicly available and have clear licensing to support reproducibility and open science.

4.2.2. VQA-MHUG as the Primary Benchmark

VQA-MHUG is the primary dataset for this thesis and provides the main evidence base for the final comparative analysis. It enables evaluation of gaze-guided preprocessing because it contains human gaze recorded during visual question answering. The same image-question pairs can also be evaluated with non-gaze preprocessing methods, making it suitable for comparing gaze-enabled and non-gaze input-reduction strategies under the same paired design.

The dataset was introduced by Sood et al. as a multimodal human gaze dataset for visual question answering. It contains gaze recordings from 49 participants and 11,970 gaze samples corresponding to 3,990 question-image pairs. The question-image pairs are drawn from the VQAv2 validation set, which also provides the human reference answers used for official VQA-style evaluation [39, 103]. Each sample consists of an image, a question, a set of 10 human reference answers, and associated gaze information.

The 3,990 samples support a thorough controlled paired comparison. The broad range of participant gaze patterns makes the evaluation of gaze-based preprocessing techniques more generalizable. For each sample with multiple participant recordings, one recording is selected at random using a fixed seed of 12345, ensuring variation in gaze patterns. This avoids a systematic participant-specific bias that would arise if the same participant were always selected, while still making the chosen trace reproducible. VQA-style soft accuracy provides a standard, interpretable measure of answer correctness, and the dataset's CC BY-NC-SA 4.0 license supports reproducibility and open science.

4.2.3. DriVQA as a Domain-Specific Case Study

DriVQA is included as a small case-study dataset for real-world visual question answering in driving scenarios. The dataset was introduced by Rekanar et al. and combines images from the NuImages dataset with questions, participant answers, gaze plots, and heatmaps collected during an eye-tracking experiment [89]. The gaze data were recorded using a Tobii Pro X3-120 eye tracker, and the dataset contains responses from 34 participants with at least two years of driving experience.

The motivation for including DriVQA is practical rather than statistical. Driving scenes provide a concrete real-use setting in which fast visual interpretation can matter, making DriVQA useful as a small external check on whether techniques that perform well on the main benchmark still behave plausibly on safety-relevant real-world images.

DriVQA is deliberately treated as a limited case study. The dataset contains 24 driving-scenario questions and 66-68 associated answers across two answer sets in the original release. The prepared DriVQA evaluation keeps the 24 questions and combines the answers from both test sets into one reference record per question. This gives each sample a distribution of approximately 66-68 human answers after merging the available participant responses. The LLM answer is therefore evaluated against the full available human-answer distribution rather than against a single canonical answer. This size is too small to support strong general statistical claims, so DriVQA results are interpreted descriptively. They are used to provide qualitative and case-study evidence about real-world driving scenarios, not to replace the main VQA-MHUG benchmark or to rank preprocessing techniques definitively. The dataset's public availability and clear licensing (CC BY-NC-SA 4.0 license) support reproducibility and open science.

4.2.4. VOILA-A as an Open-Ended Interaction Extension

VOILA-A is included as a further extension because exact-match or soft VQA accuracy does not fully capture the quality of assistive AR interaction. In practical settings, users expect responses grounded in what they are looking at. VOILA-A was introduced to align vision-language models with user gaze in realistic assistance scenarios [114].

The VOILA-A benchmark consists of two complementary resources. **VOILA-COCO** is a large automatically annotated dataset built from COCO images and trace data from Localized Narratives, where mouse traces serve as a scalable proxy for gaze. **VOILA-GAZE** is a smaller benchmark collected using an actual gaze-tracking device [114]. This thesis uses only **VOILA-COCO**, because **VOILA-GAZE** has not been publicly released for use.

VOILA-A has a different purpose from VQA-MHUG and DriVQA. It tests whether preprocessing techniques that appear efficient under short-answer evaluation still preserve the information needed for helpful, grounded open-ended responses — which matters for AR assistive settings where exact answer matching is rarely the right metric.

Figure 4.2: Example VOILA-A candidate motivating manual sample selection. The dataset reference answer is generic, while visual comparison shows that a more detailed modern multimodal-model answer can be more specific, useful, and accurate. Reference answer: "the plates contain various food items ready to be served at a dinner party", Model answer: "the plates contain roasted chicken on a bed of greens or vegetables".

VOILA-COCO is automatically annotated and contains many samples that are vague, ambiguous, or unsuitable for open-ended evaluation. Manual curation was therefore needed to select 500 usable examples for the experiment. Candidate samples were first selected randomly using a fixed seed of 12345 and then inspected together with their image, gaze trace, question, and reference answer. A sample was kept only when it passed explicit usability checks: the image file had to be available and readable; the trace had to map to the image coordinate frame; the question had to refer to visible image content rather than to missing context; the reference answer had to be non-empty and compatible with the image; and the sample had to support judging without relying on information outside the image, caption, question, trace, and reference fields. Candidates were excluded when any required component was missing. In total, 709 candidate samples were inspected, of which 500 were retained and 209 were discarded. The retained subset is stored as a fixed selection manifest in `VoilaASelectionAlgorithm/selected/prepared.json`; this file, rather than the original VOILA-COCO order, defines the VOILA-A evaluation set used in the thesis. Selection was completed before comparing preprocessing outputs, so inclusion decisions were based on sample usability rather than on the performance of any preprocessing technique.

The curation was necessary because VOILA-A evaluates open-ended response quality. Some samples had questions or reference answers too vague to support reliable comparison, a problem made worse by how quickly multimodal LLMs have improved since the dataset was released: current models routinely give more specific visual descriptions than the original reference answers.

Figure 4.2 illustrates this problem. For the question "What is on the plates in front of the woman?", the reference answer states "the plates contain various food items ready to be served at a dinner party". A modern multimodal model instead answers "the plates contain roasted chicken on a bed of greens or vegetables". Visual comparison shows that the latter answer is not only more specific and useful, but also more accurate than the dataset reference answer. Samples with this kind of weak reference signal motivated the manual selection procedure.

4.3. Experimental Setup

4.3.1. Model Selection

Model selection was guided by the intended deployment context of the study. The thesis investigates preprocessing techniques for AR-oriented multimodal inference, where response time is a central system requirement. The selected model must provide sufficient visual question answering accuracy, but it must also operate within a latency range that is plausible for interactive use. For this reason, model selection is treated as a latency–accuracy trade-off rather than as a pure accuracy-ranking problem.

A preliminary screening experiment was conducted on 100 VQA-MHUG samples using the no-preprocessing baseline. The screening covered 13 multimodal models from three providers and measured both end-to-end

latency and VQA-style answer accuracy explained in chapter 2. This screening is used to justify the model configuration for the controlled preprocessing experiments.

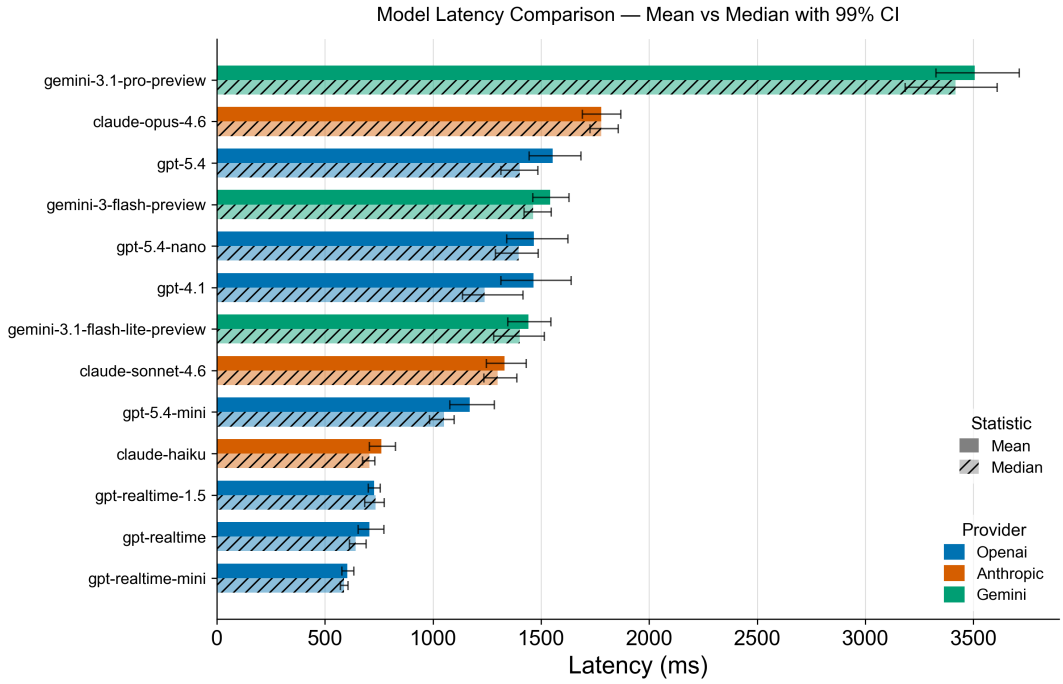


Figure 4.3: Preliminary latency screening across candidate multimodal models on 100 VQA-MHUG samples using the no-preprocessing baseline. The figure reports central tendency and variance in end-to-end latency and is used to support model selection for the main preprocessing experiments. The realtime model family achieves end-to-end latency clearly below 700 ms, while all REST API models exceed 1000 ms; this separation makes the realtime configuration the only sub-second option among the tested candidates.

As shown in Figure 4.3, the realtime models achieved substantially lower latency than the REST API models. This result is expected, as realtime models are explicitly optimized for low-latency interaction over websockets. Long response delays would reduce usability in the AR scenarios this thesis targets, making the realtime model family the most deployment-relevant choice for the main preprocessing experiments.

The accuracy screening in Figure 4.4 shows the corresponding performance trade-off. The best-performing candidate model reached 84% VQA-style accuracy, but had an average latency of roughly 3500 ms in the preliminary latency screening. In contrast, `gpt-realtime-mini` reached approximately 70% VQA-style accuracy while responding in roughly 600 ms. Although this involves a reduction in accuracy, the latency difference is large enough to make `gpt-realtime-mini` the more appropriate primary model for an AR-oriented study. `Gpt-realtime-mini` contributes to the goal of the thesis, to evaluate whether preprocessing can improve the efficiency of a realistic low-latency multimodal pipeline while preserving acceptable answer quality.

In addition to the primary realtime experiments, `gpt-5.4` is also evaluated on the VQA-MHUG dataset. `gpt-5.4` provided a useful middle ground between the realtime configuration and the highest-latency, highest-accuracy candidate: it achieved approximately 81% VQA-style accuracy with an average latency of roughly 1550 ms. Including `gpt-5.4` therefore provides insight into how the same preprocessing techniques behave under a stronger REST API model with higher accuracy but less favorable latency for interactive deployment.

The `gpt-realtime-mini` experiments form the main AR-oriented evaluation, because they measure preprocessing effects in a low-latency model configuration. The `gpt-5.4` experiments provide a REST API comparison point, allowing the thesis to examine whether preprocessing effects are specific to realtime serving or also appear in a more conventional request-response model. Additional provider-level experiments are included to test whether the relative conclusions about preprocessing remain broadly consistent across different multimodal platforms, but they are interpreted as robustness checks rather than as a full cross-provider benchmark.

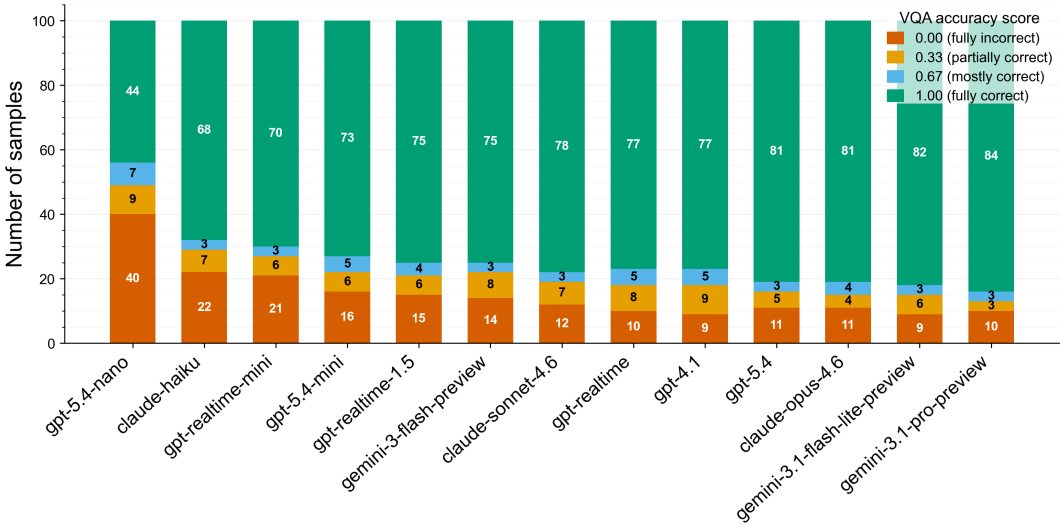


Figure 4.4: Preliminary VQA-MHUG accuracy screening across candidate multimodal models on 100 shared samples using the no-preprocessing baseline. The stacked bars show the distribution of VQA-style soft accuracy components per model. The accuracy gap between the realtime and highest-accuracy REST configurations is modest relative to the latency gap shown in Figure 4.3, supporting the selection of `gpt-realtime-mini` as the primary model for AR-oriented evaluation.

The Anthropic and Gemini configurations were selected for this robustness role because they provide contemporary commercial multimodal APIs with image input, documented token or usage accounting, and different provider-side image-handling and serving stacks from OpenAI. Anthropic Claude Sonnet was included as a strong non-OpenAI multimodal REST model with mature image understanding support, while Gemini Flash was included as a lower-latency Google model family intended for fast multimodal generation. These models were not chosen to replace the primary realtime condition; they were chosen to test whether the broad preprocessing patterns remain visible when the same 500-sample VQA-MHUG subset is evaluated through different provider APIs, pricing rules, and token-accounting conventions.

The specific versions—`claude-sonnet-4-6` and `gemini-3-flash-preview`—were identified from the official Anthropic and Google model documentation consulted on 4 May 2026, which listed these as the recommended production multimodal models for each provider at that date [4, 34]. Their performance in the preliminary screening was consistent with documented capabilities: both achieved accuracy and latency within expected ranges on the 100-sample VQA-MHUG subset. `gemini-3-flash-lite-preview` also appeared in the preliminary screening and achieved marginally higher accuracy and lower latency than `gemini-3-flash-preview`; however, since the Gemini condition functions as a robustness check rather than a primary benchmark, `gemini-3-flash-preview` was retained to match the documented production recommendation at the time of selection, rather than post-hoc optimising on 100 screening samples.

4.3.2. Variables

The experiment follows a structured variable design to isolate the effect of preprocessing from other sources of variation. The independent variable is the preprocessing technique applied to each image before it is sent to the model; this is the lever the study turns to address RQ1 and RQ2. The dependent variables capture the outcomes of interest: efficiency metrics (latency, payload size, token usage, cost) for RQ1, and task accuracy for RQ2. RQ3 then combines these two dimensions to evaluate system-level trade-offs. Controlled variables—dataset, prompt, model configuration, and hardware—are held constant so that observed differences in the dependent variables can be attributed to the preprocessing choice rather than to incidental variation elsewhere in the pipeline.

Independent Variables

Twelve techniques across five categories are evaluated, with their implementation described in subsection 4.4.2:

Category	**Techniques**
Baseline	`Baseline_NoOp`

Global reduction	`Downsampler_100x100`
	`Grayscale`
	`Jpeg_Q85`
	`WebP_Lossy_Q85`
	`GlobalThumbOnly_28x28`

Gaze-guided ROI	`GazeRoi_NoGlobalThumb` `GazeRoi_WithGlobalThumb`
Saliency-guided ROI	`SalientRoi_NoGlobalThumb` `SalientRoi_WithGlobalThumb`
Object-guided ROI	`YoloV12SalientRoi` `YoloV12SalientRoi_WithGlobalThumb`

Table 4.1: Overview of the evaluated preprocessing techniques grouped by category.

Model provider is treated as a secondary robustness factor rather than as a primary independent variable. The primary experiments use OpenAI models as the main reference point, while selected experiments are replicated with Gemini (Google) and Claude (Anthropic) to examine whether preprocessing trends remain broadly consistent across model providers.

Dependent Variables

Preprocessing can affect the pipeline at several distinct points, each captured by a separate variable. The metrics are described in detail in subsection 4.3.3; the main categories are:

payload preprocessing metrics, which quantify how the raw image input is transformed before it is sent to the model API.

latency metrics, which quantify how long the system takes to produce a usable result.

token and cost metrics, which quantify model-side usage and monetary impact.

task-performance metrics, which measure whether preprocessing changes output quality.

Controlled Variables

The following conditions are held constant across all runs:

- Image-question sample
- Implementation of each image preprocessing method
- Cloud model version and inference configuration
- System prompt and output-format instructions
- Execution machine and local software environment
- Request structure, serialization procedure, and transport protocol
- Answer normalization and scoring pipeline
- Logging, aggregation, and post-processing procedure

The paired design ensures that each technique is evaluated on the same underlying samples under the same

evaluation procedure.

Some additional factors can only be controlled partially. The local execution environment was kept consistent across runs, but complete control over network conditions, transient server-side load, provider updates, and other provider-internal effects is not possible in a cloud-based benchmarking study. These factors are treated as residual sources of measurement noise and addressed through repeated measurements, variance-aware analysis, and explicit reporting of uncertainty.

4.3.3. Evaluation Metrics

To answer RQ1 and RQ2 the evaluation uses a multi-objective metric set to compare all image preprocessing techniques against `Baseline_NoOp`.

Payload preprocessing metrics describe how the raw image input is transformed before it is sent to the model API. Table 4.2 lists all payload preprocessing metrics and their definitions. These metrics are needed to answer part of RQ1, specifically the question "to what extent do different image preprocessing techniques reduce payload size in a cloud-based multimodal inference pipeline". The metrics below capture both pixel dimensions and the encoded file size across multiple steps in the preprocessing pipeline, making it possible to analyse which steps drive size reduction and how that reduction translates into the actual transmitted payload. The metrics also capture the number of image items sent to the API, which is relevant for multi-image techniques that send multiple crops or thumbnails.

Table 4.2: Payload preprocessing metrics logged per sample.

Metric	Description
`original_pixels_w`, `original_pixels_h`	Width and height of the raw input image in pixels.
`processed_pixels_w`, `processed_pixels_h`	Width and height of the image after preprocessing.
`processed_pixels_total`	Total pixel count of the processed image (w × h); summed across all payloads for multi-image methods.
`payload_image_count`	Number of image items transmitted to the API (e.g. ROI crops plus a global thumbnail).
`roi_count`	Number of regions of interest selected by an ROI-based method.
`original_binary_bytes`	Encoded file size of the raw input image in bytes.
`processed_binary_bytes`	Encoded file size of the processed image(s) in bytes.
`binary_size_ratio`	`processed_binary_bytes/original_binary_bytes`; values below 1 indicate size reduction, values above 1 indicate size increase.
`base64_chars_total`	Total number of base64 characters across all transmitted image payloads.
`base64_bytes_utf8_total`	UTF-8 byte length of the combined base64-encoded image data.
`data_url_bytes_utf8_total`	UTF-8 byte length of the full data-URL strings (base64 data plus `` prefix).
`json_bytes_utf8`	UTF-8 byte length of the complete serialized JSON request body sent to the API.

Latency metrics quantify how long the system takes to produce a usable result. These metrics are needed to answer RQ1: To what extent do different image preprocessing techniques reduce end-to-end latency in a cloud-based multimodal inference pipeline, and what local preprocessing overhead do they introduce?. Table 4.3 lists all latency metrics logged per sample. Latency is analyzed at two levels: as an API-level end-to-end measure and as a broader pipeline measure that also includes local preprocessing overhead.

Table 4.3: Latency metrics recorded per sample. Pipeline metrics span the full local-plus-cloud interval; API metrics cover the cloud interaction only; diagnostic metrics decompose the API interval into observable client-side phases.

Field	Scope	Description
Pipeline metrics (REST and Realtime)		
`preprocess_ms`	Local	Time to apply the selected preprocessing technique (resize, compress, grayscale, ROI, saliency, gaze crop, or detector crop). Measured locally before the API call.
`end_to_end_ms`	API	Time from the start of request transmission to receipt of the final response token. Covers network transfer, provider-side processing, and streamed response delivery.
`total_pipeline_ms`	Pipeline	Sum of `preprocess_ms` and `end_to_end_ms`; the measured per-sample interval from preprocessing start to response completion.
Diagnostic stage metrics (REST)		
`client_send_ms`	API stage	Time to transmit the request payload. Derived from `client_send_end_perf_ns` − `client_send_start_perf_ns`, converted to milliseconds.
`send_end_to_first_response_ms`	API stage	Wait time from end of client send to the first observed response event. Derived from `first_response_perf_ns` and `client_send_end_perf_ns`.
`request_start_to_first_response_ms`	API stage	Combined time from request start to first response; equals `client_send_ms` plus `send_end_to_first_response_ms`.
`first_response_to_done_ms`	API stage	Time from the first response event to receipt of the complete streamed response.
Diagnostic stage metrics (Realtime only)		
`realtime_request_create_ms`	API stage	Time to create and send the per-request event within the session.
`realtime_send_end_to_first_server_event_ms`	API stage	Time from end of request creation to the first server-sent event.
`realtime_first_server_event_to_first_token_ms`	API stage	Time from the first server event to the first output token.
`realtime_first_token_to_done_ms`	API stage	Time from the first output token to completion of the streamed response.

Figure 4.5 illustrates how the client-observable latency fields relate to the local preprocessing step, the API interaction, and the measured pipeline interval. The API-level latency captures the externally visible duration of the cloud interaction, while the total pipeline latency captures the combined effect of local preprocessing and cloud interaction.

For Realtime runs, the more detailed event fields are used where available. As described in section 2.4, the Realtime API exposes additional per-request stages that REST does not.

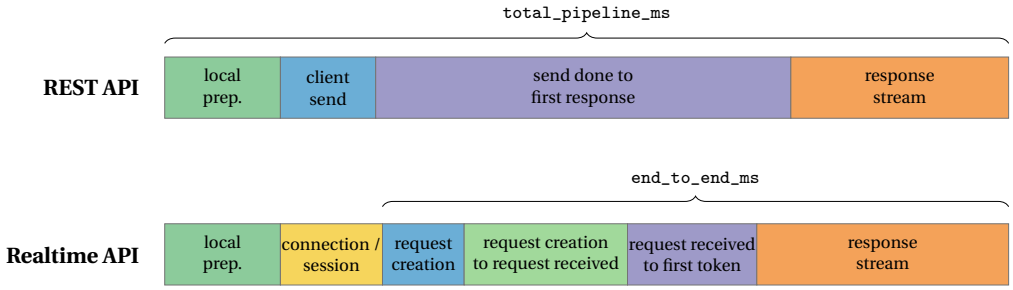


Figure 4.5: Client-observable latency decomposition used in the thesis. The full horizontal distance represents `total_pipeline_ms`, while the cloud interaction after local preprocessing represents `end_to_end_ms`. Segment widths are schematic rather than empirical.

Token and cost metrics capture the provider-side computational and monetary implications of preprocessing. These metrics aim to answer RQ1: To what extent do different image preprocessing techniques reduce token usage, monetary cost in a cloud-based multimodal inference pipeline? Table 4.4 lists the token-related fields logged per sample and their provider-side sources. These fields are recorded directly from the provider's API response where possible, rather than being estimated from local measurements or post-hoc calculations. This approach ensures that the token usage metrics reflect the provider's own accounting and billing conventions.

Cached tokens are retained as a separate field rather than merged into ordinary input tokens. Caching is expected to be zero or negligible for REST-style independent requests, but Realtime sessions can expose cached-token behaviour. The Realtime run is evaluated as a stateless sample-level interaction. A single connection is reused to eliminate connection setup costs. The provider can therefore report cached input tokens due to the reusable request context. These cached tokens are therefore retained and included in the cost calculation instead of being discarded. When cached tokens are present, they are interpreted as a subset of `input_tokens`, not as additional input tokens. The formulas used to derive per-request cost from these token fields and rates

are given in section A.1.

Table 4.4: Logged token metric fields.

Field	Description
`input_tokens`	Gross input token count for the request, including any cached tokens.
`text_input_tokens`	Input tokens attributed to the text portion of the prompt.
`image_input_tokens`	Input tokens attributed to the image portion of the prompt.
`cached_tokens`	Cached input tokens, treated as a subset of `input_tokens` rather than additional tokens.
`output_tokens`	Token count for the generated output.
`total_tokens`	Aggregate token total for the request, retained for usage analysis.
`total_cost_usd`	Estimated request cost in USD, computed from the token fields above and the per-provider rates in Table 4.5. The derivation is given in section A.1.

Table 4.5 lists the rates used for the main model configurations. The OpenAI rates are consistent with the OpenAI pricing and model documentation accessed on May 7, 2026 for the corresponding realtime-mini text rates [79, 78, 76]. The Gemini and Anthropic rates are likewise based on the corresponding provider pricing and token-accounting documentation [35, 33, 37, 5, 6].

Table 4.5: Token cost rates used for experiment cost estimation. Accessed on May 7, 2026.

Model	Text input (USD / 1M)	Image input (USD / 1M)	Cached input (USD / 1M)	Output (USD / 1M)
`gpt-realtime-mini`	0.60	0.80	0.06	2.40
`gpt-5.4-2026-03-05`	2.50	2.50	0.25	15.00
`gemini-3-flash-preview`	0.50	0.50	0.05	3.00
`claude-sonnet-4-6`	3.00	3.00	×	15.00

Task-Performance Metrics measure whether preprocessing changes output quality by using evaluation techniques. Evaluation is conducted in a dataset-appropriate manner, because the datasets used in this thesis represent different multimodal task formulations and therefore require different performance measures. Task-performance metrics answer RQ2: To what extent do different image preprocessing techniques affect visual question answering accuracy relative to a no-preprocessing baseline? and RQ3: Which preprocessing techniques achieve the highest overall suitability across deployment scenarios?.

Table 4.6 summarises the evaluation technique applied to each dataset. A complete listing of the metric fields produced by each technique is provided in section A.2.

Table 4.6: Evaluation technique applied per dataset.

Dataset	**Evaluation technique**	**Scope**
VQA-MHUG	VQA-style soft accuracy	Per sample
DriVQA	LLM judge: human-distribution rubric	Per sample; absolute and pairwise
VOILA-A	LLM judge: open-ended response quality	Per sample; absolute and pairwise

VQA-style soft accuracy is used for the **VQA-MHUG** dataset as the main quantitative measure of answer correctness, as defined in section 2.5. Each model answer is normalized using the same answer-normalization pipeline as the ground-truth answers and compared against the ten human reference answers for that sample.

The **DriVQA** case study uses a rubric based on the one shown in section 2.5. The rubric criteria, their weights, and descriptions are summarised in Table 4.7. The judge uses `gpt-4.1-2025-04-14` as the evaluator model. The evaluator also records diagnostic fields such as `major_error_type` and `short_reason`. Pairwise comparison fields against `Baseline_NoOp` include `drivqa_pairwise_winner_vs_baseline`, `confidence`, and `reason`.

Table 4.7: DriVQA rubric criteria, weights, and descriptions. Each criterion is scored on a 0–1 scale; the overall score is a weighted composite of the four criteria. Minor valid alignment is a diagnostic field and does not contribute to the overall score.

Criterion	Weight	Scale	Description
Weighted match score	50%	0–1	Agreement with the full distribution of human answers, giving more credit to answers that match common human responses.
Majority alignment	25%	0–1	Whether the answer agrees with the dominant human interpretation of the driving scene.
Ambiguity handling	15%	0–1	Whether the answer remains reasonable when human responses indicate uncertainty or multiple plausible interpretations.
Contradiction rate	10%[†]	0–1	Whether the answer contradicts the available human-answer distribution or visible scene interpretation (enters as 1 − rate).
Minor valid alignment	—	0–1	Whether the answer is compatible with a less frequent but valid human answer (diagnostic only).
Overall score	100%	0–1	Weighted composite of the four criteria above.

[†] Enters the formula as 1.0 − contradiction_rate, so lower contradiction yields a higher contribution.

For the **VOILA-A** extension, each candidate answer is scored on the four criteria summarised in Table 4.8. The VOILA-A judge uses `gpt-4.1-2025-04-14` as the judge model. In addition to numeric scores, the judge assigns a qualitative evidence-status label indicating whether the answer is supported, partially supported, contradicted, unsupported by extra detail, or impossible to assess from the available context. It also records a major error type where applicable, such as incompleteness, vagueness, hallucination, wrong object, wrong count, wrong attribute, or spatial error.

To reduce positional bias, each comparison is performed twice for both **VOILA-A** and **DriVQA**: once with the baseline shown as Answer A and once with the baseline shown as Answer B. The pairwise result is considered stable only when both orderings lead to the same interpreted winner; otherwise, it is marked as `unstable`.

Table 4.8: VOILA-A rubric criteria, weights, and descriptions. Each criterion is scored on a 0–5 scale; the overall score is the weighted sum divided by 5, normalising the result to 0–1.

Criterion	Weight	Scale	Description
Evidence consistency / accuracy	40%	0–5	Whether the answer agrees with the verified reference answer and the available scene context.
Coverage	30%	0–5	Whether the answer includes the important information needed to address the question.
Detail quality	20%	0–5	Whether the answer provides useful, relevant, and precise detail beyond a minimal response.
Unsupported claim control	10%	0–5	Whether the answer avoids hallucinated, unverifiable, or over-specific visual claims.
Overall score	100%	0–1	Weighted sum of the four criteria divided by 5.

4.4. Experimental Procedure

The experimental procedure unfolds in three phases. First, the instrumentation and execution environment are established: API configurations, timing boundaries, hardware, and software dependencies are fixed before any samples are processed, so that all measurements are collected under consistent and documented conditions. Second, the raw datasets are converted into a unified experiment-ready format and the preprocessing techniques are instantiated; completing this preparation before any model requests are issued ensures that the same inputs are reused across all preprocessing conditions within each experiment block, which is a precondition for the paired comparison. Third, the trial execution phase iterates over each sample

and preprocessing condition, dispatches requests to the cloud model, and records the response together with all timing, token, and accuracy metrics.

4.4.1. Instrumentation and Environment

The API Configurations consist of four cloud multimodal model configurations. These configurations are listed in Table 4.9.

Table 4.9: Model and API configurations used in the main preprocessing experiments.

Condition	Model identifier	API mode	Output settings
OpenAI Realtime	`gpt-realtime-mini`	OpenAI Realtime WebSocket, streamed text events	Text output only; no explicit temperature, top-p, maximum-output-token, or response-format setting
OpenAI REST	`gpt-5.4-2026-03-05`	OpenAI Responses REST API, streaming enabled	No explicit temperature, top-p, maximum-output-token, or response-format setting
Gemini REST	`gemini-3-flash-preview`	Gemini REST API, streaming enabled	`max_output_tokens=128`; no explicit temperature or top-p setting
Anthropic REST	`claude-sonnet-4-6`	Anthropic REST API, non-streaming request-response call	`max_tokens=128`; no explicit temperature or top-p setting

Table 4.10 summarises the log directories and execution windows for all runs. These dates are reported because commercial multimodal APIs can change over time, making the results time-dependent measurements of the tested configurations.

Table 4.10: Experiment log overview: directories and execution windows for each run.

Run	Log directory	Execution period
OpenAI Realtime Mini	`main_openai_realtime_mini`	2026-04-17 to 2026-04-22
OpenAI GPT-5.4 REST	`main_openai_gpt54_rest`	2026-04-15 to 2026-04-22
Anthropic Claude Sonnet REST	`main_anthropic_claude_sonnet_rest`	2026-04-28
Gemini Flash REST	`main_google_gemini_flash_3_rest`	2026-04-29
VOILA sample selection	`main_openai_realtime_mini_voila_selection_algorithm`	2026-05-06
DriVQA	`main_openai_realtime_mini_drivqa`	2026-05-06
Model-latency screening	`models_latency_experiment`	2026-05-04

All latency measurements are collected on the client side using Python monotonic performance counters. The recorded timings therefore represent externally observable client-side behavior rather than provider-internal inference time. Figure 4.6 shows exactly where each timer starts and stops for both the REST and Realtime APIs.

The latency boundary used in the reported metrics is narrower than a complete deployed AR user journey. It starts at the local preprocessing timer for a prepared dataset sample and ends when the model response has

been received by the client. Dataset enumeration, sample selection, and any image loading outside the preprocessing timer are therefore outside `total_pipeline_ms`; image decoding or encoding work is included only when it occurs inside the measured preprocessing or API interval. The metric also excludes camera exposure, sensor readout, operating-system camera buffering, speech recognition, user-intent detection, gaze calibration, display rendering, audio playback, retry attempts, backoff sleeps, quota waits, and one-time application startup or connection setup costs unless those costs occur inside the measured per-request API interval. Consequently, `total_pipeline_ms` should be read as a reproducible per-sample preprocessing-plus-cloud metric, not as a fully deployed user-facing latency measurement.

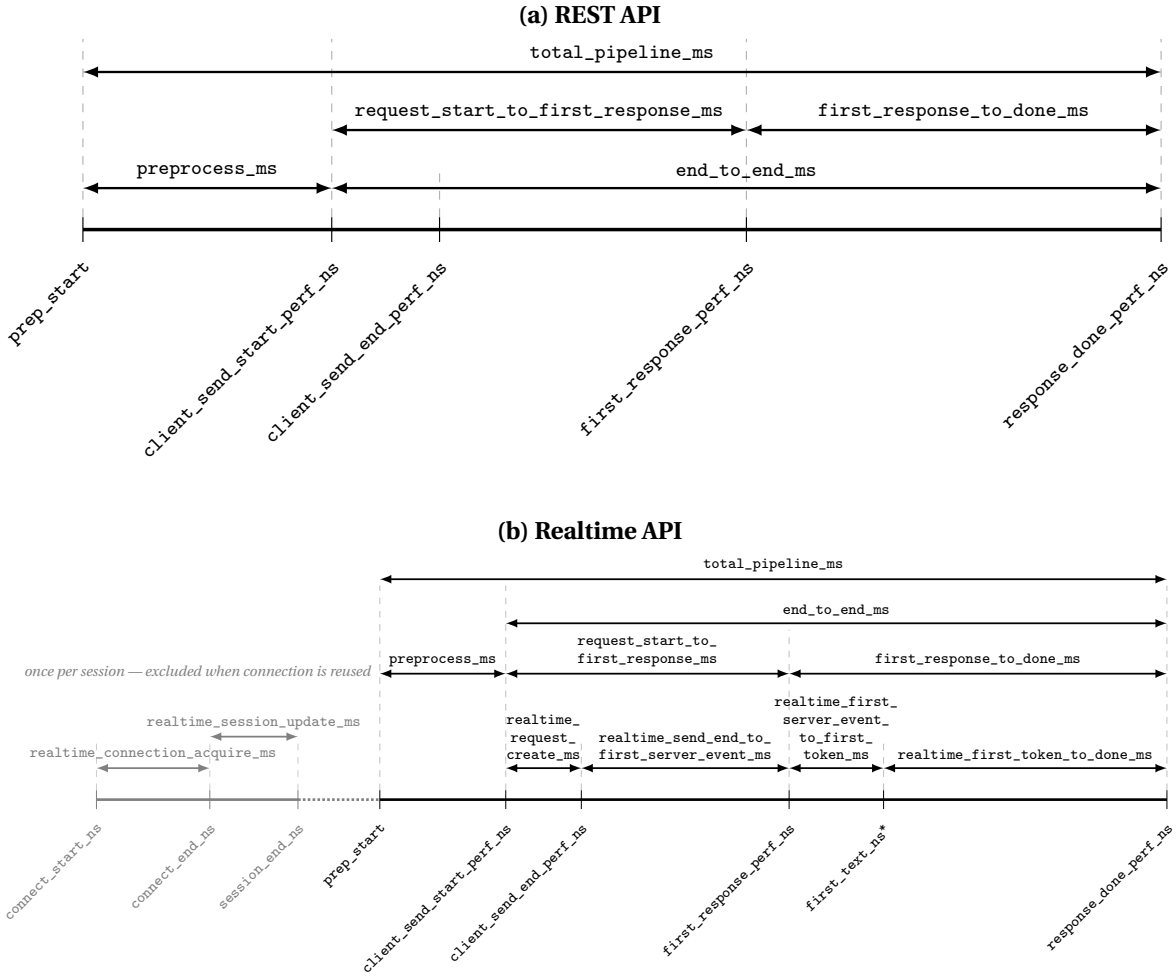


Figure 4.6: Client-side monotonic timer boundaries for REST (a) and Realtime (b) API measurements. Each labelled span is a logged metric computed from `time.perf_counter_ns()` samples; `prep_start` uses `time.perf_counter()` and its elapsed duration is `preprocess_ms`. In (b), grey segments show per-session WebSocket setup costs (`realtime_connection_acquire_ms`, `realtime_session_update_ms`) that are logged separately and excluded from per-request timing when the connection is reused (`realtime_connection_reused` = 1). *`first_text_ns` is an internal implementation variable used to compute `realtime_first_server_event_to_first_token_ms` and `realtime_first_token_to_done_ms`.

Table 4.11 lists the API response field paths from which each token metric is read. Where a provider does not expose a field natively, the value is derived or set to a fixed fallback.

All code, preprocessing implementations, experiment logs, and analysis scripts are released publicly as VQABench at `https://github.com/Hvanhuynegem/VQABench`, so that the full measurement pipeline can be reproduced and extended.

All experiments were conducted on a fixed local machine to improve the consistency of runtime measurements across samples and preprocessing conditions. The measurements reported in this thesis were obtained on an MSI laptop with an Intel Core i7-8750H CPU running at 2.20 GHz, 16 GB RAM, and a 64-bit

Table 4.11: Provider API field paths from which each token metric is populated. *Computed* means the value is derived rather than read directly; *set to* indicates a fixed fallback used when the provider does not expose that split.

Metric field	OpenAI	Gemini	Anthropic
`input_tokens`	`usage.input_tokens`	`usage_metadata.prompt_token_count`	`usage.input_tokens`
`output_tokens`	`usage.output_tokens`	`usage_metadata.candidates_token_count`	`usage.output_tokens`
`total_tokens`	`usage.total_tokens`	`usage_metadata.total_token_count`	*computed*: `input` + `output`
`cached_tokens`	sub-field of `usage.input_tokens_details`[a]	`usage_metadata.cached_content_token_count`	`usage.cache_read_input_tokens`
`text_input_tokens`	sub-field of `usage.input_tokens_details`[a]	*set to* `input_tokens`	*set to* `input_tokens`
`image_input_tokens`	sub-field of `usage.input_tokens_details`[a]	*set to* 0	*set to* 0

[a] Or the legacy field `usage.input_token_details` (singular).

Windows 11 Home operating system on an x64-based architecture. The device also includes Intel(R) UHD Graphics 630 integrated graphics and an NVIDIA GeForce GTX 1060 discrete GPU.

In the implemented object-detection preprocessor, YOLOv12 is loaded through Ultralytics from the ONNX model file `Techniques/Yolov12/yolo12n.onnx`. The code does not explicitly set an ONNX Runtime execution provider or device. The YOLO-based preprocessing results are interpreted as CPU-based ONNX inference.

The software stack used for the experiments is documented to support reproducibility of the measurement pipeline. The experiments were executed in Python 3.11.9 on Windows 11 Home. The repository dependency file is not fully pinned, so the package versions in Table 4.12 should be treated as the inspected Python environment rather than a guaranteed frozen record of the exact package state at run time.

Table 4.12: Major Python package versions observed in the inspected experiment environment.

Package	Version	Package	Version
`openai`	2.35.1	`anthropic`	0.100.0
`google-genai`	1.75.0	`httpx`	0.28.1
`websockets`	16.0	`websocket-client`	1.9.0
`onnx`	1.21.0	`onnxruntime`	1.25.1
`ultralytics`	8.4.47	`Pillow`	12.2.0
`numpy`	2.4.4	`pandas`	3.0.2
`matplotlib`	3.10.9		

The main cloud experiments were run from the Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2628 Delft, the Netherlands. No VPN was used. Unless explicitly stated otherwise, runs were executed over the building Ethernet connection, with nominal 1000 Mbps upload and 1000 Mbps download capacity. Additional network-condition experiments were performed over Ethernet, Wi-Fi, and 4G using `gpt-realtime-mini`, with 1200 logged rows for each network condition on 2026-04-22. These network-condition runs are analyzed separately and are not treated as the default condition for the main preprocessing experiments.

Requests were issued sequentially by the experiment loops, model by model and sample by sample, rather than concurrently. Retry handling is implemented in the experiment code, but the repository logs do not preserve console retry warnings. The model-latency screening log contains no failed rows and no stored error messages.

4.4.2. Data Preparation

To support paired comparison across preprocessing techniques, all datasets are converted into a unified experiment-ready format before execution. For each dataset, the preparation pipeline aligns the visual input, the question, the relevant ground-truth or reference-answer information, and gaze-related data where available. The output is a prepared representation that can be consumed consistently by the preprocessing and evaluation pipeline.

```
[
  {
    "id": "",
    "question": "",
    "answers": ["..."],
    "image_path": "",
    "gaze_points_px": [
      {
        "x": ...,
        "y": ...
      },
      ...
    ]
  },
  ...
]
```

This section also specifies the preprocessing techniques compared in the study. The general motivation for these technique families is discussed in the chapter 3. The purpose here is to define the concrete experimental conditions used in the pipeline. The evaluated techniques were selected to cover both **global reduction methods** and **selective region-preserving methods**. To illustrate the visual effects of these techniques, Figure 4.7 shows an example input image after each preprocessing method is applied. The original image (a) is from the VQA-MHUG dataset.

Baseline_NoOp. The baseline condition forwards the original image without modification. The image is still wrapped in the same API request structure as the other conditions, including Base64/data-URL serialization where required by the provider API. It is the reference condition against which all other preprocessing methods are compared.

Downsampler_100x100. This method uniformly reduces the full image to a fixed resolution of 100×100 pixels. It was implemented with the Pillow library, using function Image.resize(). The chosen target size is treated as an experimental configuration choice rather than a universally optimal setting.

Grayscale. This method converts the image from RGB to grayscale while keeping the remainder of the transmission pipeline unchanged. It was implemented with the Pillow library, using function ImageOps.grayscale().

Jpeg_Q85. This method applies lossy JPEG compression with quality level 85. It was implemented with the Pillow library, using function Image.save().

WebP_Lossy_Q85. This method applies lossy WebP compression with quality level 85. It was implemented with the Pillow library, using function Image.save().

GlobalThumbOnly_28x28. This method reduces the full image to an extreme low-resolution thumbnail of 28×28 pixels. It was implemented with the Pillow library, using function Image.resize().

The remaining techniques are **selective region-preserving methods**. These methods attempt to retain high-resolution detail only in one or more selected regions of interest. Two structural variants are considered. The **NoGlobalThumb** variants transmit only the selected region or regions. The **WithGlobalThumb** variants additionally transmit a 28×28-pixel thumbnail of the entire image to preserve coarse global context.

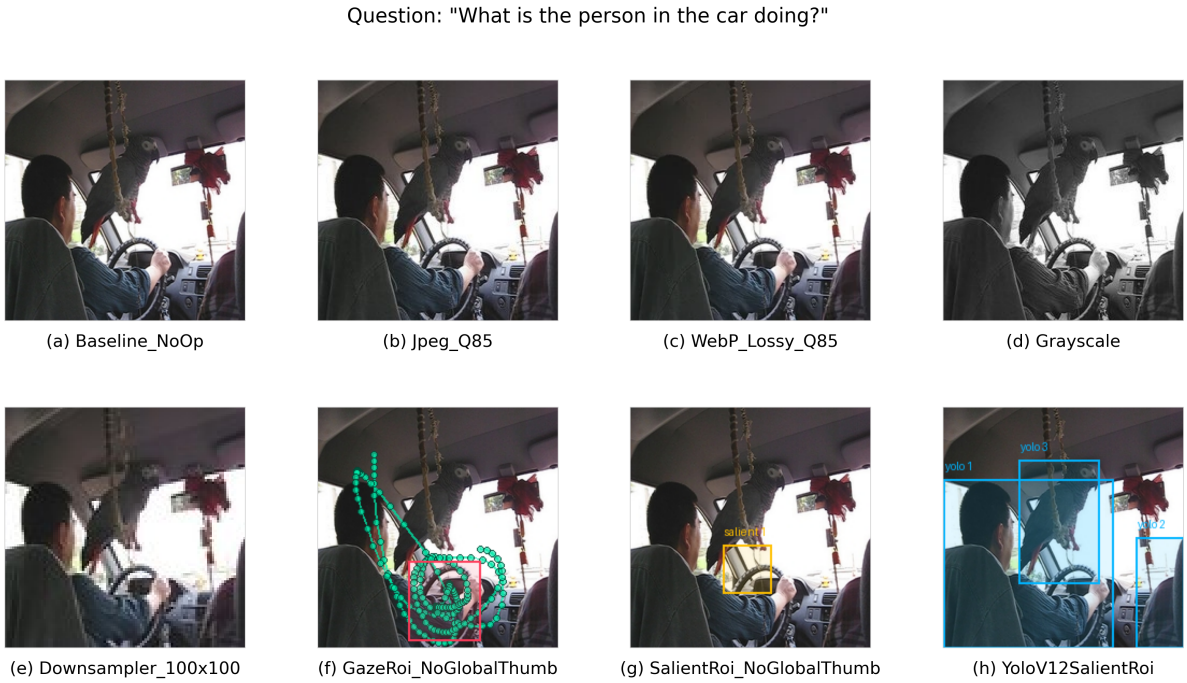


Figure 4.7: Visual comparison of the evaluated image preprocessing techniques applied to an example input from the VQA-MHUG dataset. Aggressive global reduction methods (Downsampler, GlobalThumb) produce visibly degraded images, while ROI-based methods retain full-resolution detail in the attended region at the cost of discarding context outside the crop.

GazeRoi_NoGlobalThumb and GazeRoi_WithGlobalThumb. These methods use gaze information to define a region of interest around the attended area. The gaze trace used for a sample is selected during dataset preparation. When multiple participant gaze recordings are available for the same image-question pair, the implementation randomly selects one recording using the fixed seed 12345. The same selected trace is then reused for all preprocessing techniques on that sample.

The gaze-recording window is treated as concurrent with the user's spoken question, so the main added latency of the gaze-based methods is the local preprocessing time rather than the gaze-collection interval. The timing data supporting this assumption, and two deployment strategies for realising this concurrency in a live system are described in section A.4.

The conversion from gaze points to an ROI follows the gaze-driven cropping principle described by Chen and Qi [18]. The raw gaze points are mapped onto the image plane, clipped to the valid image bounds, expanded into a continuous attention heatmap using Gaussian smoothing, and converted into an axis-aligned bounding box around the most attended region. In the reported configuration, the gaze ROI method uses $\rho = 0.5$, a minimum relative ROI size of 0.2, Gaussian smoothing with $\sigma = 15.0$, ROI detection size 64, global thumbnail size 28×28, and JPEG quality 85. The selected box is expanded when needed to satisfy the minimum relative size, clipped again to the image boundary, cropped from the original-resolution image, and then encoded as the high-resolution ROI used by the gaze-based preprocessing variants. The `WithGlobalThumb` variant adds the 28×28-pixel full-image thumbnail so that the model receives both local detail and coarse global scene context. If the selected trace cannot produce a valid ROI, the sample is treated as an invalid preprocessing case rather than silently replaced by a different gaze trace.

SalientRoi_NoGlobalThumb and SalientRoi_WithGlobalThumb. These methods use the spectral residual saliency approach introduced by Hou and Zhang [41] to identify visually prominent regions without relying on human gaze. The image is first resized to a spectral analysis size of 64 pixels, following the scale used in the original paper. The method then computes the Fourier spectrum, represents the amplitude component in log-spectrum form, and estimates the redundant spectral structure with a local average filter. The average filter size is 3×3, and the spectral residual is obtained by subtracting this local spectral average from the log spectrum. The residual is then combined with the original phase spectrum and transformed back into the

spatial domain to produce a saliency map, which is smoothed with a Gaussian filter with $\sigma = 8.0$. Candidate salient regions are selected from this map using the threshold rule $S(x) > 3.0E[S(x)]$, where $E[S(x)]$ is the mean saliency-map intensity. The resulting binary region is mapped back to the original image coordinate frame and represented as an axis-aligned bounding box clipped to the image boundary. No minimum relative ROI size is enforced in the reported configuration (`min_roi_rel_size` = 0.0), so the selected crop is determined directly by the thresholded saliency region and subsequent bounding-box extraction. If no above-threshold salient region is found, the fallback is the full image, encoded under the same default JPEG settings, and the row remains identifiable through its ROI-count and payload fields.

YoloV12SalientRoi and YoloV12SalientRoi_WithGlobalThumb. These methods use YOLOv12 object detections to identify semantically meaningful entities and convert them into one or more transmitted regions. The implementation uses the ONNX model `Preprocessing\Yolov12\yolo12n.onnx`, input size 640, confidence threshold 0.25, IoU threshold 0.45, global thumbnail size 28×28, and JPEG quality 85. Detector boxes are mapped from detector input coordinates back to original image coordinates, clipped to the image boundary, cropped from the original-resolution image, and sent as separate image items. No maximum number of detections is imposed, so the number of transmitted ROIs can vary by image. If no detection satisfies the confidence and IoU filtering rules, the fallback is the full image, encoded under the same default JPEG settings, rather than dropping the sample.

4.4.3. Trial Execution and Controls

For each selected image-question sample, the core execution pipeline proceeds as shown in Figure 4.8. First, the original image and associated question are loaded from the prepared dataset representation. Second, one preprocessing technique is selected and applied to the image. Third, the resulting processed representation is encoded and serialized into the request format used by the cloud multimodal API. Fourth, the request is transmitted and the response stream or response object is collected. Fifth, the final textual answer and all available timing, usage, and cost metadata are logged. Sixth, the answer is evaluated using the metric appropriate for the dataset.

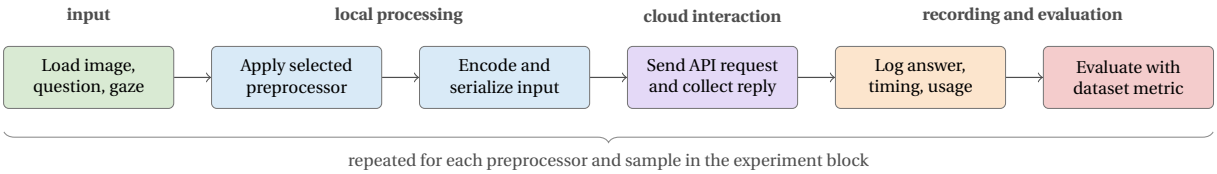


Figure 4.8: Per-sample execution pipeline used in the experimental procedure. The same conceptual flow is used across experiment blocks, while the preprocessor, model/API wrapper, prompt format, and evaluation module vary according to the dataset and condition.

Table 4.13 summarizes the final experiment blocks and sample coverage.

All random seeds used in the experiment pipeline are set to 12345. Dataset subset order is not produced by shuffling. Instead, "dataset-loader order" refers to the stable order in which the loader yields records from the relevant `prepared.json` file. VQA-MHUG reads `data/VQA-MHUG-data/prepared.json`, DriVQA reads `data/DriVQAdataset/prepared.json`, VOILA-A reads `data/voila-a-data/prepared.json`, and the VOILA-A selection set reads `VoilaASelectionAlgorithm/selected/prepared.json`. These loaders preserve JSON-array order, apart from any skipped records caused by missing images or invalid traces.

The request runners include retry handling for transient failures. The outer per-sample retry limit is eight attempts. The API clients also include up to five internal attempts with a fixed three-second internal retry delay. The outer retry wait uses provider-specified retry-after information where available; otherwise it applies exponential backoff according to $0.5 \cdot 2^r$ seconds for retry index r, capped at 30 seconds. Retryable failures include rate limits, request timeouts, connection errors, HTTP-like status codes 408, 409, 429, 500, 502, 503, and 504, and transient server-side error messages.

Preprocessing failures and final API failures are printed during execution but are not stored as metric rows. Retry attempts are also not individually logged; therefore, the recorded latency reflects the successful final attempt only and does not include earlier failed attempts, timeout duration, sleep/backoff time, or quota-

Table 4.13: Final experiment blocks and logged successful rows.

Experiment block	Model/API condition	Samples	Logged rows
VQA-MHUG main realtime	`gpt-realtime-mini`	3,990	47,880 = 3,990 samples × 12 preprocessors
VQA-MHUG OpenAI REST	`gpt-5.4-2026-03-05`	3,990	47,880 = 3,990 samples × 12 preprocessors
VQA-MHUG Anthropic check	`claude-sonnet-4-6`	500	6,000 = 500 samples × 12 preprocessors
VQA-MHUG Gemini check	`gemini-3-flash-preview`	500	6,000 = 500 samples × 12 preprocessors
DriVQA case study	`gpt-realtime-mini`	24	288 = 24 samples × 12 preprocessors
VOILA-A selection set	`gpt-realtime-mini`	500	6,000 = 500 samples × 12 preprocessors
Network-condition tests	`gpt-realtime-mini`	100 per condition	1,200 rows each for Ethernet, Wi-Fi, and 4G
Repeatability experiment	`gpt-realtime-mini`	50	6,000 = 10 repeats × 12 preprocessors × 50 samples
Model-latency screening	13 candidate models	100	1,300 = 100 samples × 13 models, baseline only

wait pauses.

Each dataset uses a distinct system instruction string that shapes how the model is expected to respond. VQA-MHUG uses strict short-answer instructions (`VQA\_SHORT\_ANSWER\_INSTRUCTIONS`) that request a one- or two-word lowercase answer without a full sentence, matching the canonical short-answer format expected by the VQA soft-accuracy metric. DriVQA uses an open-response instruction string (`DRIVQA\_OPEN\_RESPONSE\_INSTRUCTIONS`) that situates the model as a visual assistant for driving scenes and asks for a single concise natural-language sentence. VOILA-A uses a similar open-response instruction (`VOILA\_OPEN\_RESPONSE\_INSTRUCTIONS`) that requests a concise natural-language sentence without forcing a short VQA-style answer. The full text of all three instruction strings is listed in section A.3.

4.5. Statistical Reporting

All confidence intervals use 2,000 bootstrap iterations at the 99% level; bootstrap quantile intervals are used for percentile summaries (p50, p90, p95, p99). Bootstrap resampling is used rather than parametric intervals because latency and payload distributions are right-skewed: parametric normality assumptions would not hold. The 99% level rather than the conventional 95% reflects the uncontrolled network and server-side variability in cloud measurements; the more conservative threshold reduces the risk of treating noise as a real preprocessing effect. Results are reported as effect sizes and confidence intervals rather than null-hypothesis significance tests because, with up to 3,990 paired samples, even negligible differences would exceed conventional significance thresholds — practical magnitude is therefore the more informative quantity.

Descriptive statistics include means, medians, standard deviations, interquartile ranges, high percentiles, and coefficients of variation. Trade-offs are visualised with scatter, bubble, and Pareto frontier plots. Both Pearson and Spearman correlations are reported for request-size associations, with residual analyses against $\log_{10}$ payload size where appropriate. Both coefficients are included because the payload–latency relationship may be monotonic but not linear, particularly near provider-side tokenisation thresholds where additional image tiles are added discontinuously; Spearman captures such monotonic structure that Pearson would underestimate.

Each preprocessing technique is compared against `Baseline_NoOp` on the same sample IDs. Efficiency changes are reported as

$$\text{change}(x) = \frac{x_{\text{baseline}} - x_{\text{method}}}{x_{\text{baseline}}} \cdot 100, \tag{4.1}$$

where positive values indicate improvement and negative values indicate degradation. Task performance is reported as the absolute accuracy change in percentage points:

$$\Delta\text{accuracy} = 100 \cdot \left(\text{accuracy}_{\text{method}} - \text{accuracy}_{\text{baseline}}\right). \tag{4.2}$$

4.6. Threats to Validity

Conclusion Validity

Because multiple techniques and metrics are compared, there is a risk of over-interpreting isolated favourable findings. This risk is mitigated through paired analysis, effect-size reporting, confidence intervals, and conservative interpretation, but it is not fully eliminated. The thesis does not make multiplicity-corrected hypothesis-test claims. Smaller experiment blocks, such as the provider robustness checks, DriVQA case study, VOILA-A subset, and repeated-run tail analyses, have lower statistical stability than the full VQA-MHUG OpenAI runs.

Temporal validity. Commercial multimodal APIs evolve rapidly. Model behavior, token accounting, API latency, caching behaviour, and pricing can change after the measurement period. The reported results should therefore be interpreted as time-dependent measurements of the models, providers, and configurations available during the thesis experiments, not as permanent properties of the platforms studied.

Internal Validity

The main internal threat is that latency and token usage may be influenced by external cloud-service behavior rather than preprocessing alone. Network jitter, server load, rate limits, and provider-side changes can introduce variability between runs. The paired design reduces some of this risk, but it cannot remove it entirely. Another internal threat is that preprocessing methods may be affected by implementation details such as ROI parameter choices, thumbnail sizes, or detector thresholds, meaning that observed results partly reflect selected parameter settings rather than the abstract method family itself.

Construct Validity

The study uses dataset-specific task-performance measures rather than a single universal construct. VQA-MHUG uses VQA soft accuracy, which rewards agreement with annotated answer sets but does not capture every form of visual understanding. DriVQA and VOILA-A depend on LLM-based judging, so their scores inherit limitations of judge prompts, reference answers, and model interpretation. Answer normalization and reference-answer coverage can also affect measured correctness. Latency, token usage, payload size, and estimated cost are useful efficiency proxies, but they do not fully represent user-perceived responsiveness, deployment cost, or system value.

External Validity

The experiments are conducted with specific datasets, hardware, model configurations, and network environments. The strongest claims are bounded to VQA-MHUG derived from VQAv2, while DriVQA is a 24-sample driving-oriented case study and VOILA-COCO is a curated subset used for VOILA-A evaluation. Provider conclusions are mainly OpenAI-primary, with selected cross-provider checks used as robustness evidence rather than exhaustive platform coverage. The results should therefore be interpreted as evidence for the tested configuration space rather than universal performance guarantees.

Ecological validity. Although the study is motivated by AR-oriented real-time use cases, it is not conducted in a live wearable deployment with user interaction. The experiments use static images, offline or sampled gaze

information, and no closed interaction loop in which a user reacts to model output. The study does not measure perceived responsiveness, wearable compute limits, battery drain, thermal behaviour, camera capture latency, or display-rendering latency. The experimental setup therefore approximates selected perception-to-model costs, but does not fully reproduce operational AR use.

5

Results

This chapter reports the results of the controlled paired comparison across all preprocessing techniques, datasets, and providers. The latency, payload, token, and cost sections address RQ1 (efficiency impact); the accuracy section addresses RQ2 (accuracy impact). A dataset and run overview precedes the metric-specific results, and the chapter closes with a statistical reliability assessment and effect-size summary.

5.1. Dataset and Run Overview

Before presenting the metric-specific results, this section summarizes the experimental runs used in the analysis. Three datasets were evaluated, each with a different role. VQA-MHUG served as the main quantitative benchmark for measuring latency, token usage, cost, and VQA-style answer accuracy. DriVQA was used as a small driving-scenario case study. VOILA-A was used as an open-ended extension to evaluate more descriptive gaze-conditioned visual-assistance responses. The datasets are therefore reported separately rather than combined into a single aggregate accuracy score.

Table 5.1 summarizes all generation, judge, and diagnostic runs. No failed or invalid responses were removed; when failures occurred, runs were continued until every planned dataset-model-technique cell reached its target sample count. Diagnostic runs were not mixed into the main latency, token, cost, or accuracy results.

Table 5.1: Overview of all experimental runs.

Dataset	Provider/model	Interface	Samples	Techniques	Calls	Notes
Main generation runs (114,048 calls total)						
VQA-MHUG	`gpt-5.4-2026-03-05`	REST	3,990	12	47,880	Full
VQA-MHUG	`gpt-realtime-mini`	Realtime	3,990	12	47,880	Full
VQA-MHUG	`claude-sonnet-4-6`	REST	500	12	6,000	Partial; first 500
VQA-MHUG	`gemini-3-flash-preview`	REST	500	12	6,000	Partial; first 500
DriVQA	`gpt-realtime-mini`	Realtime	24	12	288	Case study
VOILA-A	`gpt-realtime-mini`	Realtime	500	12	6,000	Selected subset; 709 candidates inspected, 500 retained (seed 12345)
Judge runs (17,816 calls total)						
VOILA-A	`gpt-4.1-2025-04-14`	REST	500	12	17,000	6,000 absolute + 11,000 pairwise (2 orderings per non-baseline to reduce ordering bias)
DriVQA	`gpt-4.1-2025-04-14`	REST	24	12	816	288 absolute + 528 pairwise (2 orderings per non-baseline to reduce ordering bias)
Diagnostic runs (10,900 calls total)						
VQA-MHUG	`gpt-realtime-mini`	Realtime	50	12	6,000	Variability; 10 repeated runs per technique
VQA-MHUG	`gpt-realtime-mini`	Realtime	100	12	3,600	Network conditions: Ethernet, Wi-Fi, 4G (1,200 rows each)
VQA-MHUG	13 candidate models	REST	100	1	1,300	Model-latency screening; baseline only

5.2. Latency and Payload Results (RQ1)

This section answers the latency and payload size components of RQ1. The goal is to determine whether image preprocessing reduces the practical response time of the multimodal pipeline, whether any latency gain remains after local preprocessing overhead is included, and what the effect of payload size is on this latency.

5.2.1. End-to-end Latency Results

Realtime Latency Results

This experiment asks: does image preprocessing measurably reduce end-to-end latency in the interactive Realtime pipeline, and which techniques yield the largest reductions? The answer is yes: compact methods achieve reductions of 35–40 %, with the three best-performing techniques cutting at least 270 ms off the 726 ms baseline. `gpt-realtime-mini` is the most relevant model for this question because it is the configuration closest to an interactive AR pipeline. Table 5.2 breaks down the total pipeline latency into its main components on the VQA-MHUG dataset across 3,990 samples.

Table 5.2: Latency decomposition and baseline-relative change for the `GPT-Realtime-mini` model (REALTIME API) on the VQA-MHUG dataset.

	Local pre-processing		Request creation		Request sent → first response		First response → done		Total pipeline	
Technique	ms	Δ%	ms	Δ%	ms	Δ%	ms	Δ%	ms	Δ%
GlobalThumbOnly	16.07	**+355**	1.95	**−94**	391.98	**−41**	26.78	**+22**	436	**−40**
GazeROI	15.06	**+327**	6.33	**−80**	411.70	**−38**	20.16	−9	453	**−38**
Downsampler	16.84	**+377**	2.81	**−91**	430.31	**−36**	20.09	−9	470	**−35**
SalientROI	42.80	**+1112**	1.48	**−95**	414.98	**−38**	22.11	+0	481	**−34**
JPEG Q85	13.87	**+293**	18.58	**−41**	489.00	**−27**	21.90	−1	543	**−25**
GazeROI+T	25.67	**+627**	5.28	**−83**	515.62	**−23**	25.72	**+17**	572	**−21**
Grayscale	13.25	**+275**	17.93	**−43**	523.82	**−22**	22.23	+1	577	**−21**
SalientROI+T	45.71	**+1195**	1.39	**−96**	529.70	**−21**	22.38	+2	599	−17
Baseline	3.53	0	31.51	0	669.24	0	22.04	0	726	0
WebP Q85	90.59	**+2466**	8.07	**−74**	617.89	−8	20.43	−7	737	+2
YOLOv12ROI+T	94.0	**+2563**	8.53	**−73**	635.87	−5	21.55	−2	760	+5
YOLOv12ROI	87.89	**+2390**	8.61	**−73**	923.21	**+38**	22.08	+0	1041	**+43**

The baseline total pipeline latency is 726 ms. The best-performing techniques all reduce the total pipeline latency by at least 17%. The top three techniques are global thumbnail only (436 ms, 40% reduction, 290 ms reduction), gaze ROI (453 ms, 38% reduction, 273 ms reduction), and downsampler (470 ms, 35% reduction, 256 ms reduction). The worst-performing techniques are the YOLOv12 variants, which increase latency by 5% and 43%, respectively.

The latency increase of the YOLOv12 variants can be attributed to the overhead of running the YOLOv12 detection model locally before the request is sent. With preprocessing times of 87–94 ms, these methods pay a local cost that exceeds any server-side saving from the smaller payload.

A counter-intuitive result concerns the ordering between the two YOLOv12 variants. YOLOv12ROI+T is faster in total pipeline time than YOLOv12ROI despite including an additional thumbnail-creation step. The *Data sent to first token* column reveals the source: a 290 ms difference in server-side processing time, with all other components remaining similar between the two variants. This suggests the model has substantially more difficulty processing a bare object-detection crop than the same crop supplemented by a full-image thumbnail. Whether this 290 ms server-side difference is a model-architecture effect or a transient server-load artefact cannot be determined from the measured data alone.

GlobalThumbOnly is the fastest technique because it creates a 28×28 thumbnail in approximately 16 ms, producing the smallest payload of all tested methods and reducing both upload time and model-side processing. The more task-relevant leading candidate is GazeROI: it applies gaze information to select a scene region, achieving a latency reduction comparable to GlobalThumbOnly while retaining the image content most likely to be relevant to the user's attention. GazeROI is therefore the stronger AR-native choice when gaze data is available.

REST Latency Results

To examine whether the effect of image preprocessing on latency is consistent across model types, Table 5.3 shows the latency decomposition for `GPT-5.4` under the REST API. `GPT-5.4` is the most capable model in OpenAI's lineup as of April 2026. If the relative ordering of preprocessing techniques holds across both the Realtime Mini and REST conditions, it suggests that the preprocessing effects generalise rather than being specific to a single model or API.

Table 5.3: Latency decomposition and baseline-relative change for the `GPT-5.4` model (REST API).

	Local pre-processing		Request sent → First response		First response → Done		Total pipeline	
Technique	ms	Δ%	ms	Δ%	ms	Δ%	ms	Δ%
GazeROI	6.2	**+170**	322.5	−16	553.8	−11	883	−12
Downsampler	6.7	**+191**	337.1	−12	572.4	−8	916	−9
GazeROI+T	11.4	**+396**	322.2	−16	611.3	−2	945	−6
GlobalThumbOnly	5.3	**+130**	317.5	−17	634.0	+2	957	−5
JPEG Q85	13.3	**+478**	364.7	−5	600.9	−3	979	−3
Grayscale	4.7	**+104**	347.9	−9	651.9	+5	1004	+0
Baseline	2.3	0	382.1	0	622.0	0	1006	0
SalientROI	42.7	**+1757**	481.1	**+26**	700.2	**+13**	1224	**+22**
SalientROI+T	46.5	**+1922**	491.7	**+29**	723.6	**+16**	1262	**+25**
WebP Q85	86.5	**+3661**	506.7	**+33**	734.5	**+18**	1328	**+32**
YOLOv12ROI+T	98.7	**+4191**	365.6	−4	908.6	**+46**	1373	**+36**
YOLOv12ROI	93.0	**+3943**	485.8	**+27**	812.9	**+31**	1392	**+38**

The baseline total pipeline latency was 1006 ms, a 280 ms (39%) increase compared to the Realtime experiment baseline. The best-performing techniques are the gaze ROI method (883 ms, 12% reduction, 123 ms reduction), the downsampler (916 ms, 9% reduction, 90 ms reduction), gaze ROI with thumbnail (945 ms, 6% reduction, 61 ms reduction), global thumbnail only (957 ms, 5% reduction, 49 ms reduction), and JPEG Q85 (979 ms, 3% reduction, 27 ms reduction). The worst-performing techniques are the YOLOv12 variants, which increase latency by 36% and 38%, respectively.

The total pipeline latency change is much smaller than in the Realtime experiment, with the best technique reducing latency by only 12% compared to 40% in the Realtime condition. This suggests that the preprocessing effects on server-side processing time are less pronounced for the more capable model. However, the main ordering of techniques remains similar between both experiments, with the gaze ROI method being the best performer in both cases and the YOLOv12 variants being the worst performers. This suggests that the preprocessing effects could generalise across different models and APIs, although the magnitude of the latency reduction may vary depending on the specific model and API conditions.

Two further findings require comment. First, WebP Q85 is slow on both the Realtime and REST experiments, suggesting the format is inherently more expensive for cloud-based models to process. The preprocessing times confirm this: WebP has the longest local processing cost of any tested technique, consistent with its more complex codec requiring more computational resources to encode and decode. Second, the Salient ROI methods perform much worse in the REST experiment compared to the Realtime experiment. This is consistent with the REST model requiring more visual context to process requests quickly: saliency-guided

cropping removes information the model needs to answer the question, so it takes longer to generate a response.

Latency Variability and Tail Behavior

This section asks: do the techniques with the lowest mean latency also provide the most consistent performance, or do some introduce higher tail risk? For AR use, a technique that is fast on average but occasionally stalls for seconds is less useful than a slightly slower but predictable one. The results show that GazeROI leads on both mean and tail performance in the Realtime condition, while JPEG Q85 provides the most stable coefficient of variation across both API types. A technique that has low mean latency but high variance or a long tail may not be suitable for AR applications, as it leads to unpredictable performance and poor user experience. Figure 5.1 compares the latency percentiles for the Realtime and REST experiments.

Figure 5.1: Total pipeline latency percentiles by preprocessing technique for the OpenAI REST and Realtime experiments. Each technique is summarized by p50, p90, p95, and p99 latency. Gaze ROI shows the least amount of tail behavior in the Realtime experiment. The REST experiment shows more tail behavior across all techniques.

Figure 5.1 shows that the tail behavior of the Realtime experiment is much more contained than the REST experiment, with the p99 latency being much closer to the p50 latency for most techniques. This suggests that the Realtime API provides more consistent and predictable latency than the REST API. The gaze ROI method outperforms every other technique across all percentiles in the Realtime experiment. This suggests that the gaze ROI method not only reduces mean latency but also provides a more consistent latency performance. This can be confirmed by looking at the variance comparison in Figure 5.2, where the gaze ROI method has a much lower coefficient of variation compared to most techniques in the Realtime experiment, while in the REST experiment, it has an above-median coefficient of variation compared with other techniques.

It is important to notice that JPEG Q85 has the lowest coefficient of variation across both the Realtime experiment and the REST experiment. Comparing JPEG Q85 to the baseline, it shows a variance reduction of 0.14 in the Realtime experiment and a variance reduction of 1.40 in the REST experiment, which suggests that JPEG Q85 not only reduces mean latency but also provides a more consistent latency performance compared to the baseline. This makes JPEG Q85 a strong candidate for AR applications where both low latency and consistent performance are important, especially in cases where gaze information may not be available or reliable for the gaze ROI method.

JPEG Q85 provides the most consistent latency performance in terms of global reduction methods, while the

gaze ROI method provides the most consistent latency performance in terms of selective reduction methods.

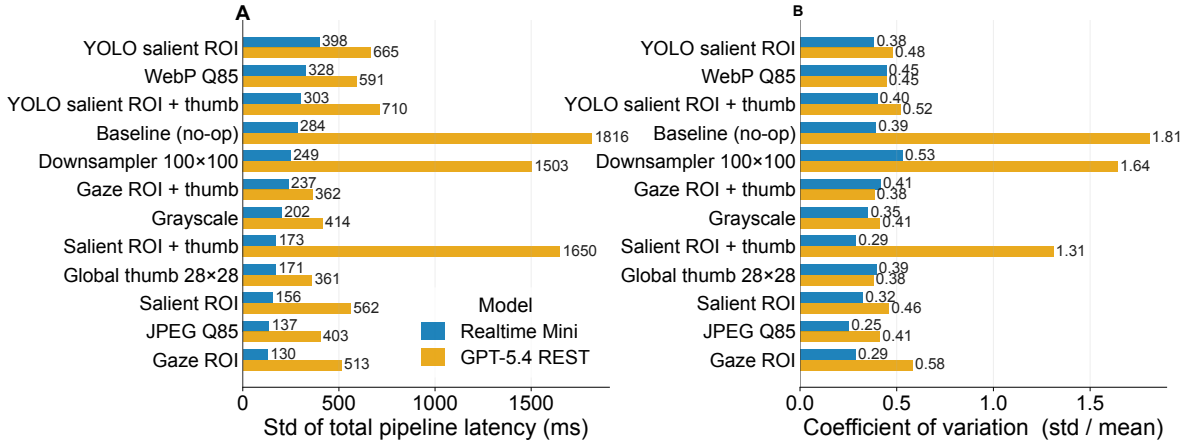


Figure 5.2: Variance in latency between Realtime and REST experiments. Gaze ROI and JPEG Q85 are highlighted as the most consistent techniques in the Realtime and REST experiments, respectively.

Cross-provider Latency Comparison

The provider comparison uses the shared 500-sample VQA-MHUG subset. This avoids comparing the full 3990-sample OpenAI REST run directly against the smaller Anthropic and Google runs. This comparison shows whether the practical latency of the pipeline is dominated by preprocessing choices or by the provider/model interface itself, and whether some providers are more sensitive to preprocessing choices than others.

Table 5.4 reports baseline total pipeline latency for the four provider/model conditions. `gpt-realtime-mini` is fastest, with a mean baseline latency of 784 ms and a p95 of 1209 ms. The OpenAI REST model `gpt-5.4-2026-03-05` is slower, with a mean of 1017 ms. The Anthropic and Google conditions are substantially slower in this baseline comparison: `claude-sonnet-4-6` averages 1633 ms, while `gemini-3-flash-preview` averages 1796 ms. The difference between the fastest and slowest baseline provider/model condition is therefore 1012 ms, which is larger than the 509 ms mean-latency spread across the selected VQA-MHUG Realtime preprocessing techniques.

Table 5.4 is sufficient for the main provider-level argument because it shows the dominant scale difference directly. Preprocessing can still move an individual condition meaningfully, as shown in the earlier within-provider analyses, but it does not erase the fact that the provider/model interface itself can shift observed latency by around one second. This reinforces the earlier REST-versus-Realtime finding that preprocessing effectiveness is implementation-dependent: reducing image bytes locally is useful only if the provider also handles the resulting image representation efficiently.

Table 5.4: Baseline total pipeline latency by provider/model on the shared 500-sample VQA-MHUG subset. All values are in milliseconds and use `Baseline_NoOp`.

Provider/model	Interface	Mean	Median	p95	Interpretation
`gpt-realtime-mini`	Realtime	784	720	1209	Fastest observed baseline pipeline
`gpt-5.4-2026-03-05`	REST	1017	894	1589	Moderate REST latency on the shared subset
`claude-sonnet-4-6`	REST	1633	1344	2640	Substantially slower baseline and heavier tail
`gemini-3-flash-preview`	REST	1796	1718	2591	Highest mean and median baseline latency

Figure 5.3 shows the mean latency change for each preprocessing technique across the four provider/model conditions. The best-performing techniques are consistent across all four conditions, with gaze ROI, global thumbnail, and downsampler techniques showing the largest latency reductions in each case. YOLOv12-based techniques and WebP Q85 show the largest latency increases across providers/models, and could be ruled out as candidates for AR deployment on latency grounds alone.

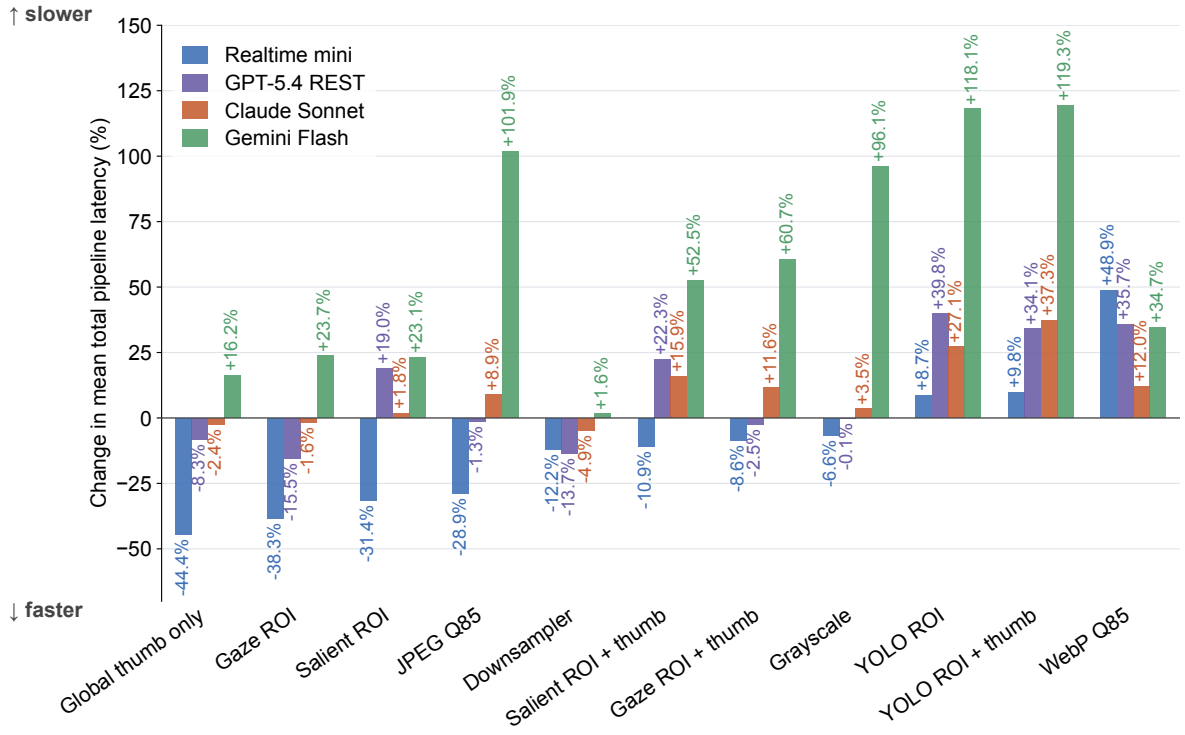


Figure 5.3: Change in mean total pipeline latency by provider/model on the shared 500-sample VQA-MHUG subset. Gaze ROI, global thumbnail, and downsampler techniques are highlighted as the best-performing techniques across providers. The provider/model interface produces a much larger latency shift than any of the preprocessing techniques.

The cross-provider comparison also reveals a structural difference between the Realtime and REST conditions. For `gpt-realtime-mini`, compact representations produce clear latency reductions: the model responds faster when given less data to process, consistent with a streaming interface optimised for low-latency throughput. The REST models — `gpt-5.4-2026-03-05`, `claude-sonnet-4-6`, and `gemini-3-flash-preview` — show a weaker relationship between payload reduction and latency saving. These models are optimised for response quality, and allocate inference effort toward producing accurate answers regardless of image resolution, so byte reduction does not translate proportionally to a faster response. For methods that discard task-relevant image content — such as the YOLOv12 ROI variants — the effect is reversed: stripping visual evidence forces the model to work harder to produce an answer, which can increase rather than decrease server-side inference time. WebP Q85 increases latency through a different mechanism: its local preprocessing cost is the highest of any tested technique at approximately 91 ms (Table 5.6), and the format may incur an additional server-side conversion penalty since the baseline input is JPEG — a provider that re-encodes WebP internally would add server-side processing time on top of this local cost.

Cross-dataset Latency Comparison

The dataset comparison uses the `gpt-realtime-mini` runs for VQA-MHUG, DriVQA, and VOILA-A. These datasets differ in visual domain and task formulation. The comparison should therefore be read as a practical dataset-and-task comparison rather than as a controlled measurement of image content alone.

Table 5.5 first anchors the comparison in the no-preprocessing baseline. DriVQA has the highest baseline mean and median latency, although its 24-sample size means that its p95 should be treated as a descriptive small-sample estimate. VQA-MHUG sits between the other datasets, with a mean of 726 ms, a median of 643 ms, and an intermediate p95 of 1352 ms. VOILA-A has the most compact baseline distribution, with the lowest mean, median, and p95 among the three dataset runs. The comparison therefore suggests that dataset and task formulation do affect Realtime latency, but the strength of this conclusion differs by dataset because VQA-MHUG contains 3,990 baseline samples, VOILA-A contains 500, and DriVQA contains only 24. section A.3 provides the prompt instructions for each dataset. VQA-MHUG has the longest and most complex

prompt, which could contribute to its higher latency.

Table 5.5: Baseline total pipeline latency by dataset for the `gpt-realtime-mini` runs. All values are in milliseconds and use `Baseline_NoOp`.

Dataset	Samples	Mean	Median	p95	Interpretation
VQA-MHUG	3990	726	643	1352	Intermediate baseline tail
DriVQA	24	863	803	1404	Highest baseline latency, but small-sample p95
VOILA-A	500	615	566	930	Most compact baseline distribution

Figure 5.4 is the main dataset-level visual because it shows both dataset differences and the effect of selected preprocessing techniques. This is more informative than a baseline-only distribution: it keeps the no-preprocessing reference visible while also showing how compact representations and ROI-style methods shift the latency distribution within each dataset. On VQA-MHUG, the selected techniques span approximately 241 ms in median latency, from `GlobalThumbOnly_28x28` at 402 ms to `Baseline_NoOp` at 643 ms. The same ordering does not transfer perfectly to the smaller datasets: the global thumbnail remains the lowest-latency option in each panel, but gaze ROI is clearly favourable for VQA-MHUG and VOILA-A while it is much more variable in the 24-sample DriVQA case study. This confirms that preprocessing can materially change response time within the same model interface, but that the effect depends on the dataset and task context. The main difference observed between the datasets is therefore the Gaze-based method performing worst on the DriVQA dataset. It has the largest range of variation in total pipeline latency and a 2250 ms p99 outlier. Gaze based preprocessing might not perform well on these questions since for most questions, global information is needed, for example, how many vehicles are there in the frame is a hard question for MLLMs to answer having only a gaze based ROI.

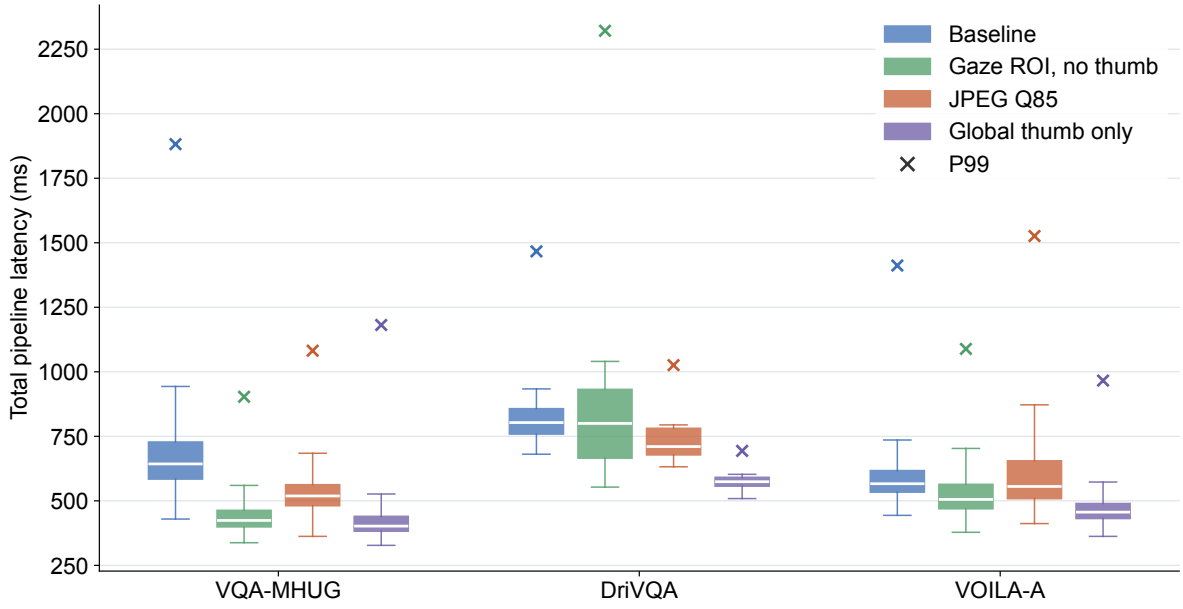


Figure 5.4: Total pipeline latency distributions for selected preprocessing techniques across the `gpt-realtime-mini` dataset runs. The figure shows global thumbnail and gaze ROI distributions being the two most favorable options.

Network Condition Effects on Latency

The network-condition experiment adds an external deployment-oriented source of variability. Figure 5.5 shows that total pipeline latency depends on the connection type, but the pattern is not simply that all wireless conditions behave alike. Ethernet produced the lowest mean latency and the tightest distribution. The 4G condition was slower than Ethernet, but it behaved more similarly to Ethernet than to Wi-Fi: its mean latency, variance, and 75th percentile remained comparatively compact, indicating that the mobile connection was reasonably stable in this experiment. Wi-Fi performed worse, with a higher and more dispersed latency

distribution. The ECDF in Figure 5.6 makes this effect more explicit for the request-sent-to-first-response interval. Ethernet reaches high completion fractions at the lowest latencies, Wi-Fi follows with a modest delay, and 4G lags further behind. The upper part of the ECDF is especially informative: above roughly the 75th percentile, the 4G curve remains comparatively linear and close in shape to Ethernet, whereas the Wi-Fi curve becomes less consistent in the final 25% of requests. These results are important because they show that preprocessing effects are embedded in a broader systems context. A method that is acceptable on Ethernet may still be usable on a stable 4G connection, but may feel less reliable under poorer wireless conditions, especially when the p95 or p99 latency rather than the mean is used as the criterion.

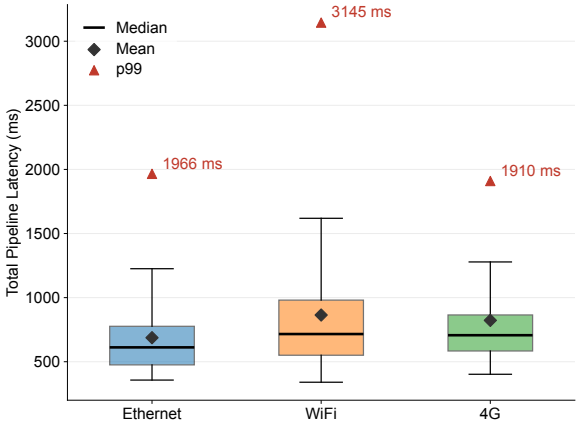


Figure 5.5: Distribution of total pipeline latency by network condition in the Realtime network-condition experiment. Ethernet shows the lowest latency and tightest distribution, 4G is slightly slower but still compact, while Wi-Fi has a higher and more dispersed latency distribution.

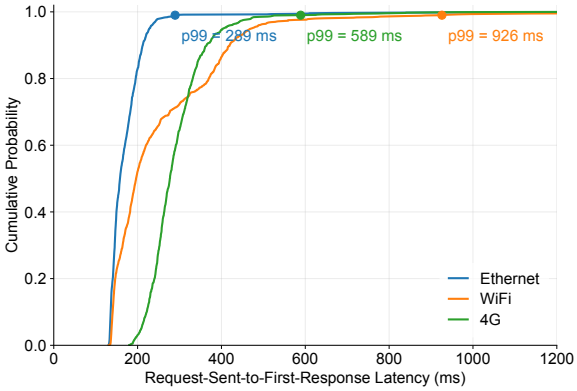


Figure 5.6: Empirical cumulative distribution of request-start-to-first-response latency by network condition. Ethernet reaches high completion fractions at the lowest latencies, 4G follows with a modest delay, and Wi-Fi lags further behind, especially in the upper percentiles.

Figure 5.5 shows that the p99 value of both Ethernet and 4G are around 1950 ms, with 4G performing slightly better than Ethernet. This suggests that the tail risk of 4G is not substantially higher than Ethernet in this experiment, which is encouraging for mobile AR applications. However, the Wi-Fi condition has a much higher p99 value of 3145 ms, indicating that the tail risk is much higher under Wi-Fi conditions. This could be due to a variety of factors, such as interference from other devices, signal strength issues, or network congestion. The high tail risk under Wi-Fi conditions suggests that it may not be a reliable option for AR applications that require low latency and consistent performance. The network results therefore suggest that SIM-based mobile connectivity can be practical for real-time AR devices. The main caution is that all cloud-based conditions retain some tail risk, and this risk increases under less favourable network conditions.

5.2.2. Local Preprocessing Overhead Results

This section answers the second part of RQ1: What local preprocessing overhead is introduced by each image preprocessing technique? The difference between end-to-end latency and total pipeline latency is determined by the local preprocessing step. Table 5.6 illustrates this local cost with the Realtime and REST preprocessing time as a percentage of the total pipeline time. Simple image transformations remained inexpensive: `Grayscale`, `Downsampler_100x100`, `Jpeg_Q85`, `GazeRoi_NoGlobalThumb`, `GazeRoi_WithGlobalThumb` and `GlobalThumbOnly_28x28` all stayed between approximately 13 and 17 ms. These values are small compared with the Realtime mean latency of 726 ms reported above in Table 5.5, which explains why the latency reductions from these methods mostly survive when moving from end-to-end latency to total pipeline latency.

One would expect the local pre-processing latency for a given technique to be device- and dataset-dependent, not API-dependent, since the computation runs entirely on the client before sending the request. However, when comparing the pre-processing latency reported in Tables 5.2 and 5.3, it can be seen that some of the techniques cost more under the Realtime WebSocket interface than under the REST interface. Table 5.6 summarises these discrepancies: lightweight techniques (GazeROI, Downsampler, GlobalThumbOnly) run 2–3× slower in the Realtime setting, whereas compute-heavy techniques (SalientROI, YOLO, WebP) show near-

parity. These overhead results are likely due to the WebSocket connection required by the Realtime API rather than the preprocessing itself.

Table 5.6: Local pre-processing latency comparison across API interfaces. The RT / REST ratio reveals techniques whose cost differs between server interfaces. A ratio deviating from 1 is explained by Websocket-related jitter. The last two columns show pre-processing time as a percentage of total pipeline latency.

Technique	**Realtime** (ms)	**REST** (ms)	**Realtime / REST**	**RT prep (%)**	**REST prep (%)**
GlobalThumbOnly	16.07	5.3	3.03	3.7	0.6
Grayscale	13.25	4.7	2.82	2.3	0.5
Downsampler	16.84	6.7	2.51	3.6	0.7
GazeROI	15.06	6.2	2.43	3.3	0.7
GazeROI+T	25.67	11.4	2.25	4.5	1.2
Baseline	3.53	2.3	1.53	0.5	0.2
WebP Q85	90.59	86.5	1.05	12.3	6.5
JPEG Q85	13.87	13.3	1.04	2.6	1.4
SalientROI	42.80	42.7	1.00	8.9	3.5
SalientROI+T	45.71	46.5	0.98	7.6	3.7
YOLOv12ROI	87.89	93.0	0.95	8.4	6.7
YOLOv12ROI+T	94.00	98.7	0.95	12.4	7.2

The RT/REST ratio difference for lightweight techniques arises from WebSocket connection overhead: the GPT Realtime API maintains a persistent WebSocket or WebRTC connection and continuously handles the realtime communication and media pipeline [82, 83]. These persistent-session responsibilities introduce additional sources of latency variability, including CPU scheduling jitter and event-loop blocking, which can contribute to latency fluctuations in real-time systems [11, 101]. For low-computational-load tasks, such as downsampling, grayscale conversion, and thumbnail generation, the preprocessing itself completes in only a few milliseconds, placing it in a regime where system-level jitter can dominate the measurement and introduce substantial variance [88, 12]. In contrast, for computationally intensive tasks, such as JPEG/WebP compression, image salient detection, or YOLO-based object detection, the runtime is dominated by the actual processing workload, and constitutes the vast majority of the measured latency [49, 59]. In this regime, the additional overhead introduced by the GPT Realtime pipeline becomes proportionally small and does not significantly affect the measured preprocessing latency.

Preprocessing time vs Payload size Relationship

Figure 5.7 shows the relationship between the mean payload size and the mean preprocessing time per preprocessing technique. `GazeRoi_NoGlobalThumb`, `GlobalThumbOnly_28x28`, and `Downsampler_100x100` again show the lowest combined overhead: their local preprocessing times are low and their payload sizes are small. The WebP and YOLO-based methods have much higher local preprocessing times, and payload sizes, which helps explain why they do not produce latency reductions despite their smaller than baseline payloads. The local overhead of these methods is high enough to offset the transmission and server-side processing benefits of the smaller payload, leading to overall latency regressions.

5.2.3. Payload Size Results

Payload size vs Latency Relationship

This section asks: how strongly does payload size predict total pipeline latency, and is byte reduction sufficient to explain the observed latency gains? Payload size reliably predicts the transmission-sensitive part of the pipeline, but not total latency. Smaller serialized image inputs reduce client-side transmission time and may reduce provider-side image handling, but the experiments show this relationship holds cleanly only for the former; local preprocessing overhead, model-side processing, and encoding format also contribute materially to total pipeline time.

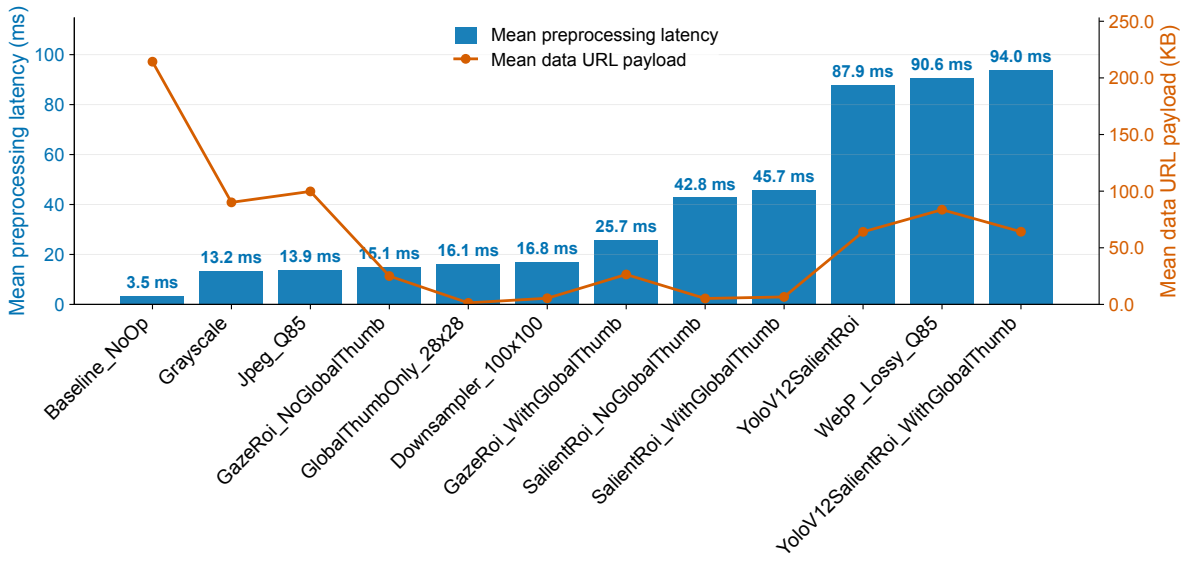


Figure 5.7: Mean local preprocessing latency and mean serialized data URL payload size by preprocessing technique for the OpenAI Realtime experiment on the VQA-MHUG dataset. There is a clear global minimum of payload size, showing that some methods achieve substantial payload reduction with only a modest local cost, while others have high local cost without producing latency benefits.

The Realtime results show a clear relationship. Figure 5.8 shows that, payload size and total pipeline latency have a moderate positive association, however, there is little correlation. Larger payloads tend to be slower, and the no-preprocessing baseline occupies the high-payload, high-latency region. Payload size therefore helps explain Realtime latency, but it does not determine it. Techniques with similar payload sizes can still differ substantially in total pipeline latency, indicating that model-side image handling, response generation, and method-specific encoding matter more.

Focusing on the most transmission-sensitive component sharpens the picture. Figure 5.9 plots payload size against the time from request-send completion to the first server event, the interval most directly exposed to transfer overhead. Here the Spearman correlation rises to ($\rho = 0.82$), while the Pearson correlation stays low at ($r = 0.14$). Larger Realtime payloads consistently wait longer for the first server event, but the relationship is not linear.

To separate the contribution of byte size from that of decoded pixel geometry, a controlled calibration varied JPEG quality at fixed pixel dimensions — so that only the serialized byte size changes while the submitted image resolution remains constant. This calibration finds $R^2 \leq 0.02$ for the three REST providers and $R^2 = 0.10$ for `gpt-realtime-mini` (see section A.7). Server-side compute therefore dominates total pipeline latency for REST interfaces: byte reduction via JPEG compression lowers transfer overhead without accelerating server-side inference.

Overall, payload reduction is useful but incomplete as an explanation of latency. The Realtime experiment provides the clearest evidence that smaller payloads reduce the transmission-sensitive part of the request, and this contributes to the strong latency performance of compact methods such as `GlobalThumbOnly_28x28`, `GazeRoi_NoGlobalThumb`, and `Jpeg_Q85`. However, total latency also depends on local preprocessing overhead, image encoding format, the number and structure of transmitted image inputs, model-side image processing, response generation, and external service conditions. For AR-oriented deployment, this means that preprocessing should be evaluated as a full pipeline strategy rather than as byte reduction alone.

Cross-dataset Payload Comparison

This section asks: does dataset-level variation in native image resolution explain the payload differences observed in the cross-dataset latency analysis? The results confirm this: DriVQA images are natively larger, producing substantially higher baseline payloads and correspondingly higher latency. The JPEG Q85 re-encoding anomaly also surfaces here, illustrating a practical concern for deployment on embedded devices

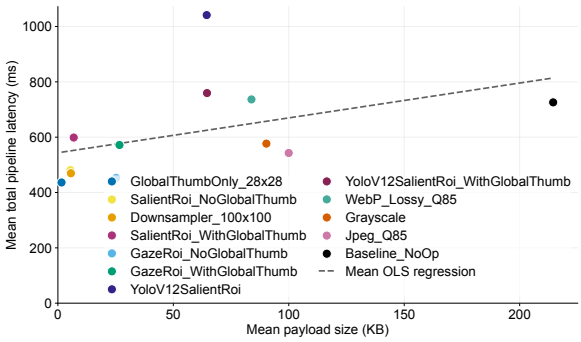


Figure 5.8: Relationship between mean serialized payload size and mean total pipeline latency by preprocessing technique for the OpenAI Realtime experiment. Payload size and total latency have a moderate positive association, but the relationship is not linear: local preprocessing overhead and model-side processing also contribute, so byte reduction alone does not determine total pipeline latency.

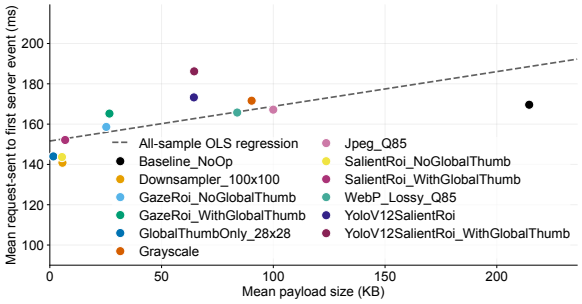


Figure 5.9: Relationship between serialized payload size and the Realtime request interval from send completion to first server event. The annotation reports Pearson and Spearman correlations for raw samples and preprocessor-level means. The strong rank correlation ($\rho = 0.82$) alongside the weak linear correlation ($r = 0.14$) shows that larger payloads consistently wait longer for the first server event, but the relationship is monotonic rather than proportional.

with already-compressed sensor outputs.

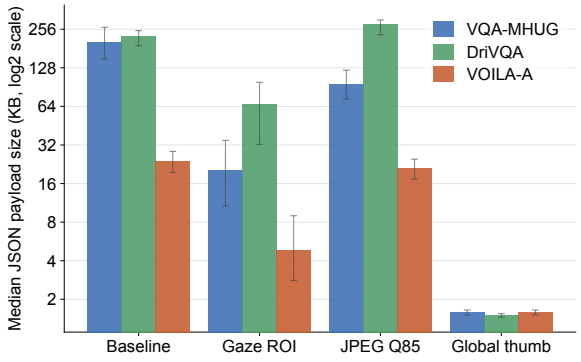


Figure 5.10: Median serialised payload size in KB ($\log_2$ scale) per dataset and preprocessing technique. DriVQA has the largest payload baseline due to its natively higher image resolution; JPEG Q85 increases payload on DriVQA because its source images are already more compressed than Q85 quality, so re-encoding at that quality raises the file size.

Figure 5.11: Bytes per pixel relative to the no-preprocessing baseline for JPEG Q85 and WebP Q85. JPEG Q85 increases bytes per pixel on already-compressed images; WebP Q85 consistently reduces bytes per pixel.

Preprocessor	bytes/pixel vs baseline
Baseline	0.131 (0%)
JPEG Q85	0.155 (+20%)
WebP Q85	0.106 (-19%)

Figure 5.12: Median native image resolution per dataset. The large resolution difference between VOILA-A and DriVQA directly drives the baseline payload differences observed in Figure 5.10.

Dataset	Median Resolution (pixels)
VOILA-A	224 × 224
VQA-MHUG	640 × 480
DriVQA	1600 × 900

Figure 5.10 shows how the payload size gets affected by each preprocessing technique across the three datasets. Every technique sees a decrease in median payload size except for JPEG Q85. It experiences a rise in median payload size compared to the baseline. This is counter-intuitive, since JPEG Q85 is a compression algorithm. The reason for this comes from the DriVQA dataset already having highly compressed JPEGs. When the preprocessor decodes them and re-encodes them at Q85, it is actually increasing the quality relative to the original, which makes the file larger. Figure 5.11 compares JPEG Q85 against WebP Q85, it can be seen that only JPEG adds more data. It is therefore important to verify in which encoding quality images are saved/retrieved from sensors on embedded devices so that no extra data or latency is added to the pipeline.

Figure 5.10 also reveals that among the datasets there seems to be less data sent for VOILA-A compared to DriVQA. This effect can also be observed in Figure 5.4: DriVQA being slower in total pipeline latency than VOILA-A. The reason for this can be attributed to the median pixel resolution as demonstrated in Figure 5.12. The median resolution per image changes drastically across the datasets.

5.3. Token Usage and Monetary Cost Results (RQ1)

This section addresses the token-usage and cost components of RQ1. Token counts indicate how preprocessing changes the amount of information reported by a provider, but are not directly comparable across providers because each uses its own tokenizer, image-token accounting rules, and pricing. Dollar cost is therefore the normalized metric for cross-provider comparison. The section first examines whether preprocessing reduces token usage and whether those reductions mirror payload-size and latency trends, then translates token differences into dollar cost.

5.3.1. Vision Token Sensitivity to Image Resolution and Aspect Ratio

The LLM service providers examined in this study follow the same three-step server-side processing pipeline on image tokenisation. It first decodes the received compressed image into a pixel grid, optionally resize it to a provider-specific target, then tokenise the result [77, 7, 36]. Because the image is decoded before tokenisation, image tokens depend on the resulting pixel grid alone, not on how the original image was compressed.

For instance, OpenAI's GPT-4o-class models resize the image to fit within a 2048^2 box with the shortest side scaled to 768 px, then tile it into 512×512 patches at 170 tokens per tile plus an 85-token base charge [77]. GPT Mini and Nano variants use 32×32 patches with a 1,536-patch budget and a model-specific multiplier. Anthropic tokenises approximately as $\lceil w \cdot h/750 \rceil$ tokens [7], where w and h are the two dimensions of the pixel grid. Gemini issues a flat 258 tokens for images whose longer side is at most 384 px; beyond that threshold, it tiles the image at 768×768 and charges 258 tokens per tile [36]. A 1024×1024 image therefore maps to roughly 765 tokens on GPT-4o, ~ 1,400 tokens on Claude, and ~ 1,032 tokens on Gemini — nearly a 2× spread for the same pixels.

The accounting is also discontinuous: shrinking an image just below a tile boundary collapses its token count, but shrinking it slightly above the boundary does not. A preprocessing technique that is favourable on one provider can be neutral on another and counterproductive on a third, and a single-provider benchmark cannot detect this.

These theoretical accounting differences were verified empirically. section A.5 reports controlled calibrations on image resolution and JPEG compression quality. The key findings are three-fold: (i) JPEG compression reduces serialized payload size but does not affect provider-reported token usage or cost for any tested model, because tokenisation operates on the decoded pixel grid; (ii) `gpt-realtime-mini` and `ge mini-3-flash-preview` report near-constant token counts across the tested resolution range, whereas `gpt-5.4-2026-03-05` and `claude-sonnet-4-6` show stepwise increases consistent with tiled accounting; and (iii) for `gpt-realtime-mini`, the token bucket is determined by decoded aspect ratio rather than pixel count or encoding format, with square images falling into a lower-cost 194-token bucket regardless of total pixel area. Consequently, downsampling can reduce token usage only when it crosses a provider-side resolution threshold, while JPEG compression can reduce upload payload and potentially latency without any reduction in token usage or cost.

5.3.2. Token and Cost Reduction Across Preprocessing Techniques

This section asks: which preprocessing techniques reduce provider-reported token usage and estimated cost, and does byte reduction predict cost reduction? The results show that byte reduction and billable cost reduction are distinct: only techniques that change the submitted visual geometry achieve meaningful cost savings, while compression-only methods (JPEG Q85, WebP Q85) leave token usage and cost near the baseline.

Token and cost results are interpreted per model condition because provider accounting is not directly comparable across platforms. The no-preprocessing baseline gives the cost of submitting the original image; each technique is then read as a change relative to that.

Figure 5.13 shows percentage changes in token usage and estimated cost against the baseline. A baseline-relative view keeps the provider-specific accounting context fixed while showing both direction and magnitude. The hatched variance bar is a log2 ratio of each technique's total-token variance to the baseline's: zero

means equal variance, positive means more variable, negative means more stable. A value of +1 is roughly twice the baseline variance; −1 is roughly half. This tells whether a technique produces a consistent token footprint or stays sensitive to image size, gaze position, salient-region size, or tiling thresholds.

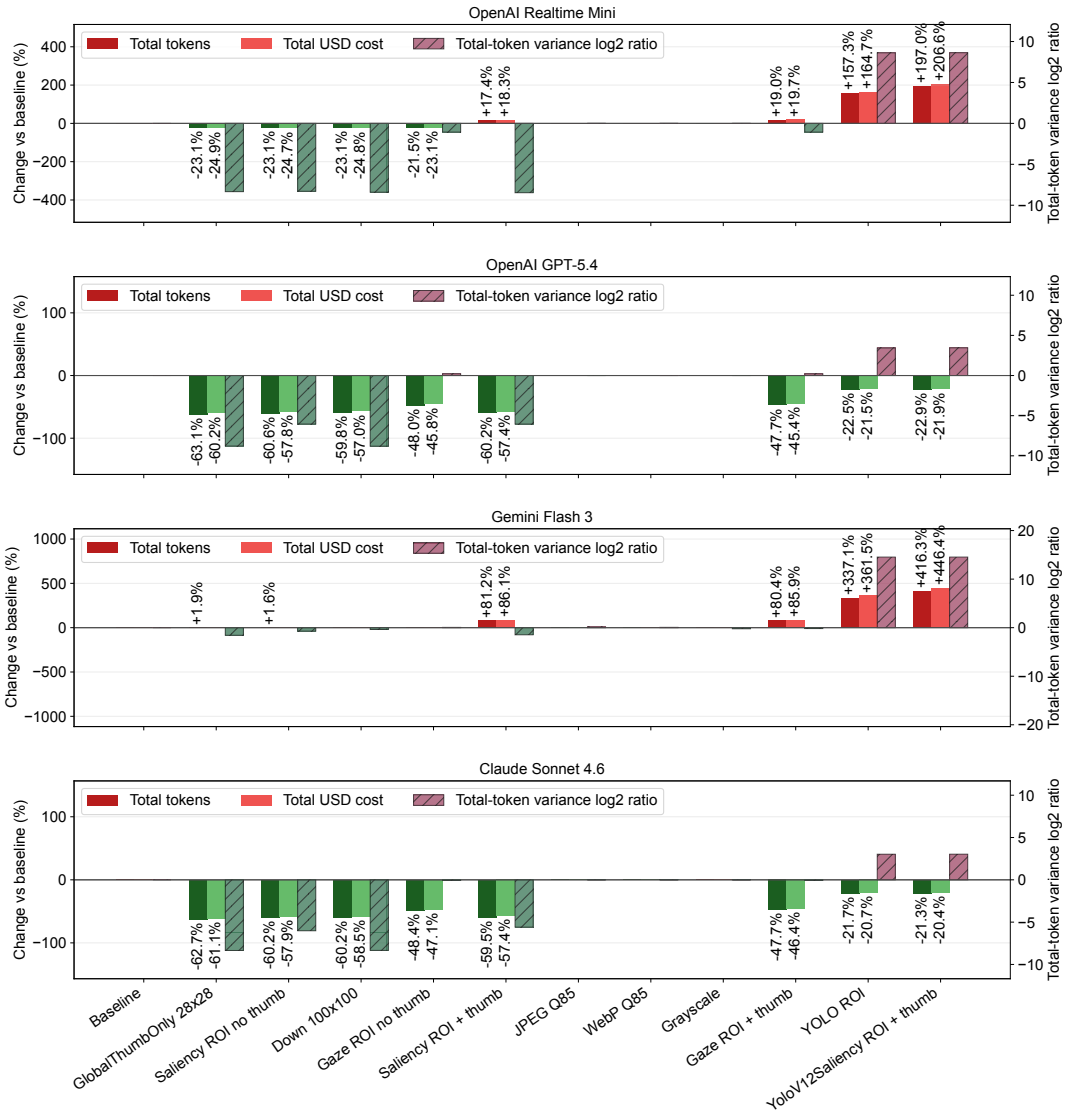


Figure 5.13: Percentage change in provider-reported token usage and estimated total request cost relative to `Baseline_NoOp` by preprocessing technique. The figure combines central percentage changes with variance shown on a log2 scale, so it captures both accounting savings and stability in one view. ROI and downsampling methods achieve the largest combined savings; JPEG Q85 and WebP Q85 reduce payload substantially but leave token usage and cost near the baseline, confirming that billable cost depends on decoded visual geometry rather than compressed file size.

Cost falls when a method changes the submitted visual structure — decoded geometry, aspect-ratio bucket, or image-part count — not when it just reduces byte size. `GlobalThumbOnly_28x28`, `Downsampler_100x100`, and the no-thumbnail ROI methods are the cheapest group in the Realtime condition; `Jpeg_Q85`, `Grayscale`, and `WebP_Lossy_Q85` stay near baseline. The downsampler calibration showed that pixel count alone did not drive token usage in the square test; the aspect-ratio comparison explains why padding a non-square image into a square can move it into a cheaper bucket. The cheapest representation is not necessarily the best one for the task, though — small thumbnails and tight crops also strip visual evidence. `Jpeg_Q85` and `WebP_Lossy_Q85` cut file size substantially but gain nothing on tokens because the provider charges for decoded image content, not compressed bytes.

ROI methods split on whether they include a global thumbnail. `GazeRoi_NoGlobalThumb` and `SalientRoi_NoGlobalThumb` crop the visual field and reduce cost when the crop crosses relevant resolution or tile

thresholds. Adding a thumbnail — `GazeRoi_WithGlobalThumb` and `SalientRoi_WithGlobalThumb` — costs more input budget in exchange for global scene context in the submitted request. Whether that context actually helps accuracy is a question for the VQA results.

Under Realtime accounting, `GlobalThumbOnly_28x28`, `Downsampler_100x100`, `GazeRoi_NoGlobalThumb`, and `SalientRoi_NoGlobalThumb` fall in the low-cost region of Figure 5.13. `Jpeg_Q85` does not — it sits near baseline and is not a cost-reduction method in this condition. Among the cheap methods, `GazeRoi_NoGlobalThumb` is the most task-specific: it narrows the submitted field using gaze-derived focus rather than a uniform downsample. In absolute terms, mean input tokens dropped from 534 to 278 for GPT-5.4 and from 598 to 308 for Claude Sonnet — roughly halving the image budget — while Realtime Mini saw a shallower reduction from 518 to 407 mean tokens, and Gemini remained near its baseline of 1 372 tokens.

5.3.3. Cost Decomposition: Input, Image, and Output Contributions

This section asks: in which billing component do preprocessing-driven cost savings appear, and do output costs remain stable across techniques? The results show that all savings are input-side effects: output cost remains small and uniform across all preprocessing techniques, confirming that reductions from visual-geometry changes do not affect response length.

Figure 5.14 decomposes mean cost per 1 000 requests into its contributing components, text input, visual input, cached input, and generated output, confirming that the savings observed for ROI and downsampling methods are entirely input-side effects. Output cost remains small and stable across all preprocessing techniques, consistent with the short-answer VQA format. Because output tokens carry a higher per-token price than input tokens, dollar cost is the more informative unit for cross-provider comparison: raw token counts cannot be directly compared across providers because pricing structures differ, whereas cost provides a common currency. The dominant variation appears in the visual input component of the Realtime condition, which falls for methods that reduce submitted image geometry. Providers that do not expose a separate image-token field report an aggregate text-input bar rather than a component split, so the decomposition is cleanest for GPT-5.4, Claude Sonnet, and Gemini. Because the same samples were used across all preprocessing techniques, text-input cost should be uniform across preprocessors; any variation in the text-input bar for Claude Sonnet and GPT-5.4 is therefore attributable to how those providers apportion image charges within their aggregate billing. For all models except Gemini Flash 3, preprocessing-driven cost reduction is achievable by selecting a technique that modifies the visual structure the provider bills against. The cached-input component visible in the Realtime condition arises from OpenAI storing prompt state across WebSocket connections, as described in section 2.4; a developer cannot disable this behaviour.

5.4. Accuracy Results (RQ2)

This section answers RQ2 by examining whether the efficiency gains reported above come at the cost of answer quality. Accuracy is evaluated relative to `Baseline_NoOp` on matched samples. As established in section 2.5, 1 percentage point is the upper bound of acceptable degradation when accuracy is the primary constraint. In deployment contexts where latency or cost rank above accuracy — such as the real-time AR use case this thesis targets — a somewhat larger drop may be acceptable if the efficiency gain is proportionally large.

5.4.1. VQA-MHUG Soft Accuracy

VQA-MHUG is the main quantitative accuracy benchmark. It uses the official VQA-style soft accuracy metric defined in section 2.5: the normalized model answer is compared with ten normalized human reference answers, producing per-sample scores of 0, 0.33, 0.67, or 1.0. The mean over samples gives the technique-level accuracy score. Because all preprocessing techniques are evaluated on matched sample IDs within a model condition, differences from `Baseline_NoOp` can be interpreted as paired changes rather than as independent group differences.

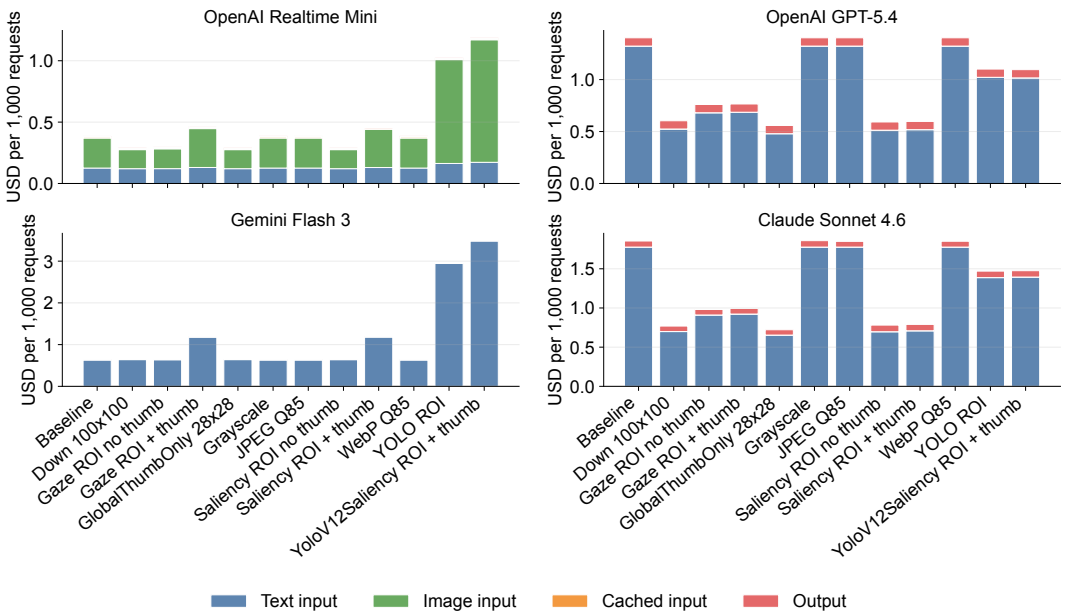


Figure 5.14: Mean cost per 1 000 requests by preprocessing technique, decomposed by billing component. The stacked bars separate text input, visual input, cached input, and generated output contributions where the provider logs expose these components; providers that do not report a separate image-token field contribute an aggregate text-input bar. All cost savings appear in the visual-input component; output cost remains small and stable across all techniques, confirming that geometry reduction does not affect response length.

Table 5.7 quantifies these differences as baseline-relative changes in percentage points with 99% bootstrap confidence intervals, sorted from least to most accuracy loss in the Realtime Mini condition. A consistent ordering emerges: compression-only techniques (`Jpeg_Q85` and `WebP_Lossy_Q85`) sit closest to the baseline across every model, gaze-based ROI methods follow as the best selective-reduction option, and thumbnail-only and saliency-only reductions occupy the low-accuracy end.

`Jpeg_Q85` and `WebP_Lossy_Q85` show no statistically meaningful accuracy loss: their 99% confidence intervals include zero across all four models, and Gemini Flash 3 trends slightly positive for both.

Gaze-based ROI methods form a consistent second tier under the Realtime condition (−3 to −4 pp), but their losses grow substantially on the full-resolution REST models (−9 to −10 pp for GPT-5.4 and Claude Sonnet 4.6). This divergence only becomes apparent when comparing across model tiers. Because the SOTA models achieve around 90% accuracy on the full image, they correctly answer nearly every sample — including those whose answer region falls outside the gaze ROI. The Realtime model is less accurate overall, so these gaze-missed samples tend to be wrong in both conditions and the gap does not surface. Figure 5.15 makes this explicit: the "Baseline correct / method wrong" segment grows sharply for gaze-based methods in the higher-accuracy model conditions, confirming that they discard image content needed for a non-trivial subset of questions.

`Downsampler_100x100` is the most inconsistent technique: its loss is moderate in the Realtime (−6.6 pp) and Gemini (−2.5 pp) conditions but severe for GPT-5.4 (−23.5 pp), indicating that fine-grained detail becomes important when the model is capable of exploiting it. Grayscale and YOLO-based variants form a middle cluster with consistent losses of −5 to −11 pp across all models.

Saliency-only and thumbnail-only methods occupy the lowest accuracy tier, with losses exceeding 20 pp in every condition — a direct consequence of discarding large portions of relevant image content.

Figure 5.15 confirms the same pattern at sample level, using a correctness threshold of soft-accuracy score greater than zero. The best methods preserve most baseline-correct cases and occasionally recover a baseline-wrong case, while the weaker methods show a progressively larger "Baseline correct / method wrong" segment. This paired breakdown isolates the source of accuracy loss: degraded methods fail on samples that the original image could answer, not because preprocessing reveals new discriminative infor-

Table 5.7: VQA-MHUG soft accuracy by preprocessing technique across four model conditions. Each cell shows absolute accuracy (top) and mean change in percentage points relative to `Baseline_NoOp` (bottom) with 99% bootstrap confidence intervals. Sorted by Realtime Mini Δpp. Techniques above the baseline row have confidence intervals that include zero in all four models. **Bold green**: CI includes zero; mild green: small confirmed loss ($|\Delta| \leq 5$ pp); orange: moderate loss; **bold orange**: large loss (>15 pp).

Technique	Realtime Mini (n=3,990) acc. / Δpp	GPT-5.4 (n=3,990) acc. / Δpp	Gemini Flash 3 (n=500) acc. / Δpp	Claude Sonnet 4.6 (n=500) acc. / Δpp
JPEG Q85	75.1% **−0.5** [−1.8, +1.0]	88.1% **−0.3** [−1.1, +0.7]	88.0% **+1.5** [−0.9, +4.3]	84.1% ≈0.0 [−2.4, +2.4]
WebP Q85	74.7% **−0.8** [−2.5, +0.6]	87.9% **−0.5** [−1.3, +0.4]	88.2% **+1.7** [−1.2, +4.0]	84.0% **−0.1** [−2.5, +2.4]
Baseline	75.6% 0	88.3% 0	86.5% 0	84.1% 0
GazeROI	72.5% −3.1 [−4.9, −1.5]	78.1% −10.2 [−11.7, −8.7]	81.6% −4.9 [−9.1, −1.3]	75.5% −8.5 [−13.2, −3.9]
GazeROI+T	72.1% −3.5 [−5.0, −1.9]	78.8% −9.5 [−11.0, −8.0]	84.3% **−2.2** [−6.1, +1.7]	74.7% −9.4 [−13.9, −5.2]
Downsampler	69.0% −6.6 [−8.5, −4.7]	64.9% **−23.5** [−25.1, −21.4]	84.1% **−2.5** [−6.2, +1.9]	67.8% **−16.3** [−21.1, −11.9]
Grayscale	68.8% −6.8 [−8.4, −5.1]	82.1% −6.2 [−7.4, −5.0]	81.7% −4.8 [−8.6, −1.4]	75.3% −8.7 [−12.5, −5.1]
YOLOv12ROI+T	68.6% −7.0 [−8.8, −5.4]	77.3% −11.0 [−12.6, −9.6]	81.5% −5.0 [−8.9, −1.0]	73.5% −10.6 [−15.0, −5.8]
YOLOv12ROI	67.4% −8.2 [−9.9, −6.2]	78.0% −10.4 [−11.9, −8.8]	80.9% −5.7 [−9.4, −2.3]	75.0% −9.1 [−13.4, −4.3]
SalientROI+T	53.5% **−22.1** [−24.3, −19.4]	53.7% **−34.6** [−36.6, −32.7]	61.3% **−25.2** [−30.7, −20.5]	52.5% **−31.5** [−37.7, −25.7]
GlobalThumbOnly	51.1% **−24.5** [−26.9, −22.7]	46.9% **−41.5** [−43.5, −39.3]	56.1% **−30.5** [−36.0, −25.0]	49.1% **−34.9** [−40.7, −29.7]
SalientROI	44.5% **−31.1** [−33.3, −29.0]	52.4% **−35.9** [−37.8, −33.9]	54.9% **−31.6** [−37.1, −26.0]	48.5% **−35.5** [−41.4, −29.7]

mation. In the Realtime condition, preprocessing does recover some baseline-wrong cases — accounting for roughly 7% of all answers — but the same methods lose more than 10% of answers in that condition, so the net effect remains negative.

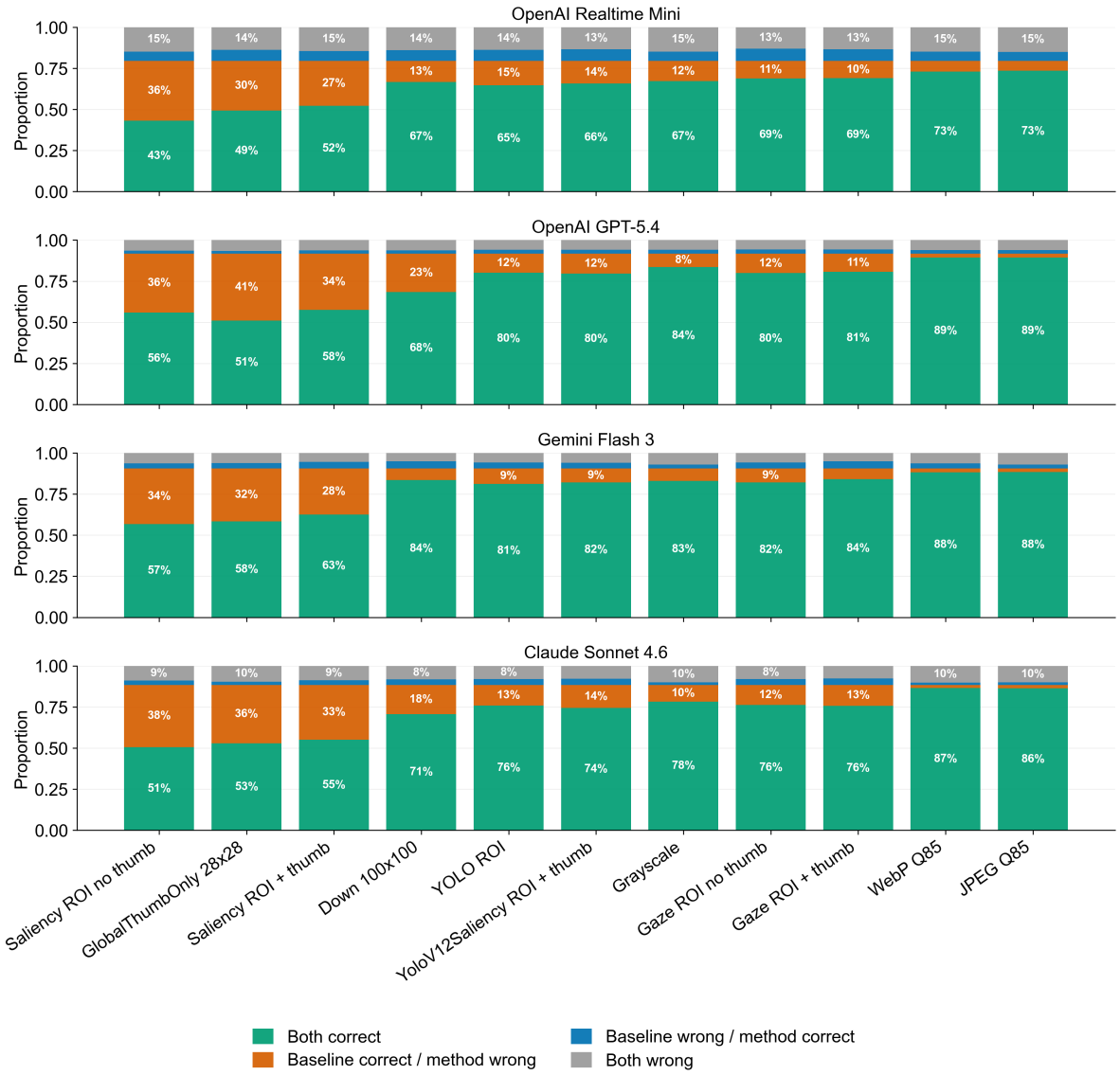


Figure 5.15: Paired sample-level correctness comparison between each preprocessing technique and `Baseline_NoOp`. The *Baseline correct / method wrong* segment grows sharply for thumbnail-only and saliency-only methods, revealing that accuracy loss stems from discarding image content the full image could answer correctly, not from any gain in discriminative information.

5.4.2. DriVQA Case-Study Accuracy

DriVQA contains only 24 driving-scenario questions and is therefore reported as a descriptive case study rather than a ranking benchmark. Each response was evaluated with an LLM judging protocol producing weighted match scores, overall scores, and contradiction rates.

The ordering in Figure 5.16 mirrors the VQA-MHUG pattern: `Jpeg_Q85` and `WebP_Lossy_Q85` sit closest to the baseline, followed by the gaze-based ROI methods and `Grayscale`. The weakest techniques—saliency-only, YOLO-based, and thumbnail-only—also show the highest contradiction rates, consistent with the larger benchmark. One exception is `Downsampler_100x100`, which scores below `GlobalThumbOnly_28x28` here, reversing their VQA-MHUG ordering; given $n = 24$ this is most plausibly sampling variation. The majority alignment and overall scores track closely across all preprocessors, confirming the judge's scores are

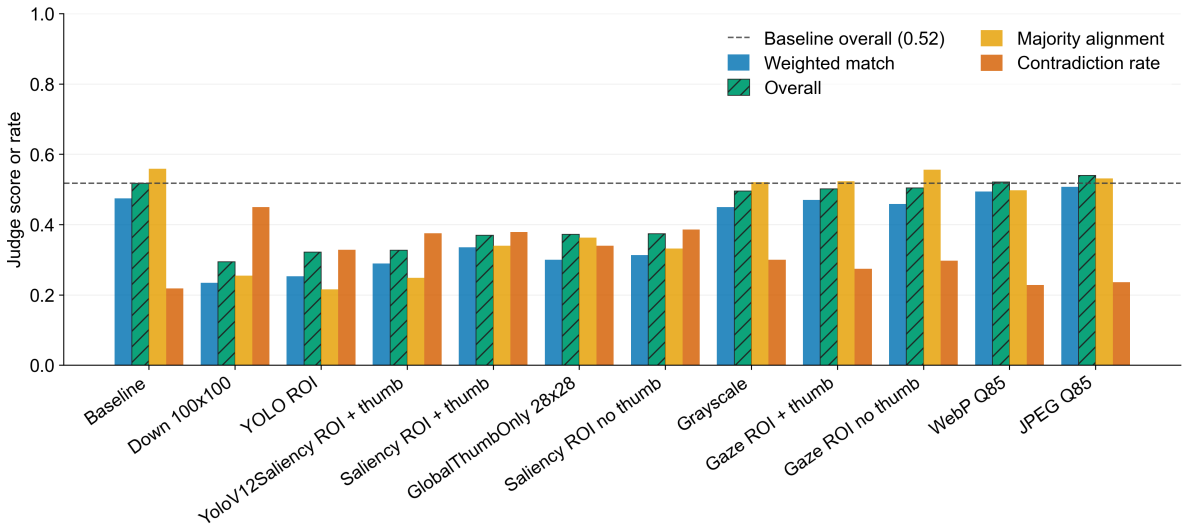


Figure 5.16: DriVQA judge-based case-study accuracy by preprocessing technique ($n = 24$, descriptive only). The ordering mirrors the VQA-MHUG pattern: compression-only methods sit closest to the baseline while saliency-only and thumbnail-only methods show the highest contradiction rates.

internally consistent. Taken together, DriVQA corroborates the main finding that moderate compression and gaze-based ROI are the most reliable choices for domain-specific driving-scenario questions. That said, the baseline itself achieves an overall score of only 52%, which falls short of what real-world deployment would require. As shown in the VQA-MHUG results, gaze-based methods only begin to fail at higher model accuracy levels, so their performance on this dataset warrants re-evaluation once stronger models are available.

5.4.3. VOILA-A Open-Ended Quality

VOILA-A is evaluated with the rubric defined in subsection 4.3.3 rather than VQA soft accuracy, because its prompts are open-ended and gaze-conditioned. The rubric produces separate scores for evidence consistency, coverage, detail quality, and unsupported-claim control, combined into a normalized overall score. These results answer a different version of RQ2: whether preprocessing preserves useful open-ended assistance quality, not only short-answer correctness.

`Jpeg_Q85` and `WebP_Lossy_Q85` preserve open-ended quality closest to the baseline; gaze-ROI methods are usable but weaker; saliency-based and thumbnail-only methods degrade quality substantially. In Figure 5.17, `Jpeg_Q85` marginally exceeds the baseline (0.745 vs. 0.738) and `WebP_Lossy_Q85` matches it exactly (0.738), confirming that compression-only preprocessing imposes no quality penalty.

`GazeRoi_NoGlobalThumb`, `GazeRoi_WithGlobalThumb`, `Downsampler_100x100`, and `Grayscale` form a usable middle cluster (0.688–0.721). One counter-intuitive result is that `Grayscale` (0.721) achieves the lowest unsupported-claim rate of all methods, including the baseline (0.280 vs. 0.326), indicating that the model produces fewer hallucinated claims when colour information is absent. The tier structure mirrors the VQA-MHUG ordering, with compression-only, gaze-based, and saliency-based methods appearing in consistent rank across both datasets, so the overall score is a reliable cross-dataset summary.

Saliency-based cropping causes the largest drop: `SalientRoi_NoGlobalThumb` scores 0.450, a 39% reduction in overall quality, with the unsupported-claim rate rising from 0.326 at baseline to 0.584.

The pairwise comparison in Figure 5.18 confirms that only compression methods genuinely close the gap with the baseline. For `Jpeg_Q85` and `WebP_Lossy_Q85`, the baseline wins only 18–19% of comparisons and ties reach 34–35%, meaning the model produces near-identical output for approximately half of samples. `SalientRoi_NoGlobalThumb` is worst by a wide margin: the baseline wins 71% of pairwise matchups. `Downsampler_100x100` warrants separate attention: despite a moderate absolute score (0.688), it shows the highest pairwise instability of any preprocessor at 30%. A 30% instability rate means the LLM judge pro-

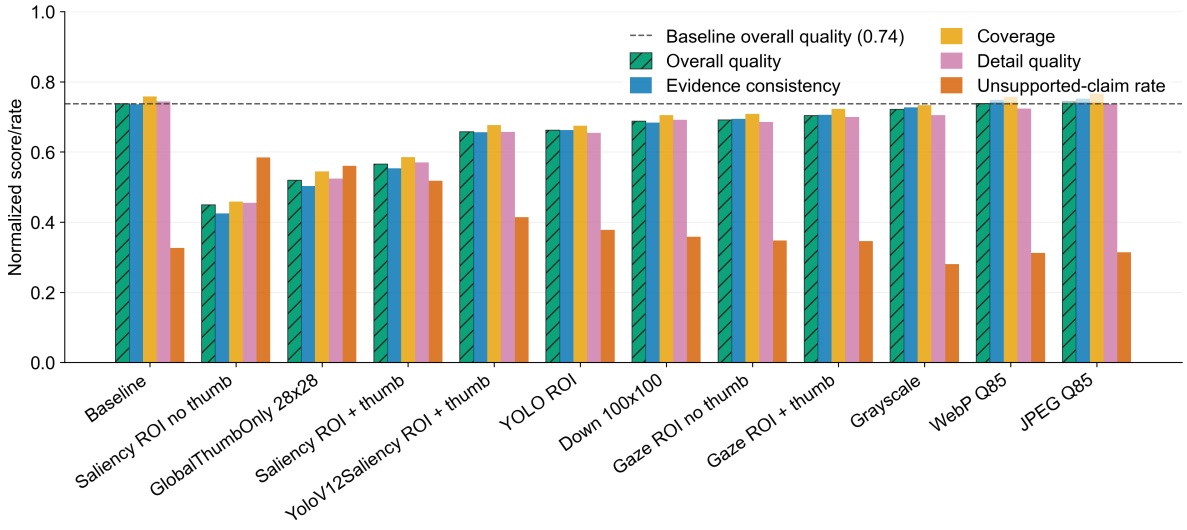


Figure 5.17: VOILA-A absolute judge quality scores by preprocessing technique. The overall score combines evidence consistency, coverage, detail quality, and unsupported-claim control. Compression-only methods match or exceed the baseline; saliency-based cropping causes the largest quality drop (overall score 0.450), with the highest unsupported-claim rate of any preprocessor.

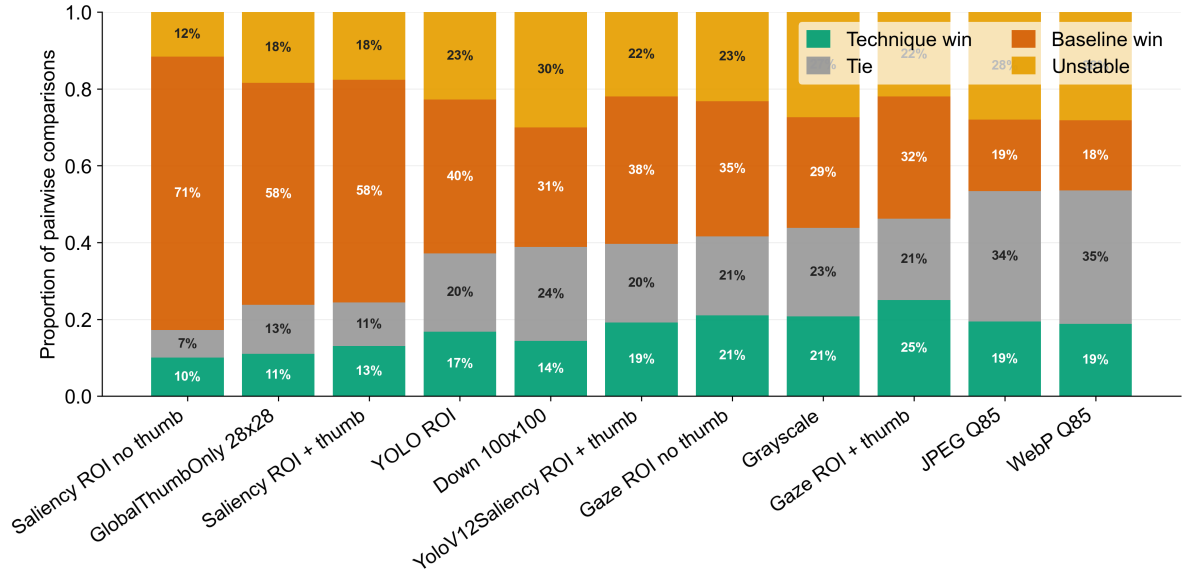


Figure 5.18: VOILA-A pairwise judge comparison against `Baseline_NoOp`. Each method is evaluated in both answer orderings to reduce positional bias. Only `Jpeg_Q85` and `WebP_Lossy_Q85` genuinely compete with the baseline; `Downsampler_100x100` shows the highest judge instability (30%) of any preprocessor.

duces an inconsistent verdict in 30% of comparisons when the answer order is reversed, indicating that the quality difference between baseline and `Downsampler_100x100` is too small for the judge to resolve reliably. This does not invalidate the absolute-score penalty (a drop of 0.050 from baseline), but it means the pairwise ranking for this method should be treated with less confidence than for methods where the judge is consistent. For AR assistance, a fluent but weakly grounded response is more harmful than a short answer that admits uncertainty; the deployable choices are therefore the compression-only methods, which keep evidence consistency, coverage, and unsupported-claim rates at baseline levels.

5.5. Statistical Reliability

5.5.1. Uncertainty and Confidence Estimates

This section assesses the evidential strength of the metric-specific findings reported above. Uncertainty is characterized through 99 % bootstrap confidence intervals, paired baseline-relative effect sizes, distribution plots, and repeated-run diagnostics, as specified in section 4.5. Effect size is preferred over p-values for the reasons stated there; no multiplicity-corrected hypothesis tests are reported. Because 12 techniques are compared across six metric dimensions, isolated favourable results carry higher false-discovery risk than findings that converge across datasets and metric types; cross-evidence consistency is therefore the primary guard against over-interpretation.

Realtime latency. The evidence here is strongest. The best-performing techniques—`GlobalThumbOnly_28x28`, `GazeRoi_NoGlobalThumb`, and `Downsampler_100x100`—reduce mean total pipeline latency by 35–40 % relative to `Baseline_NoOp`, with 99 % confidence intervals well below zero. Crucially, the effect is not confined to the mean: the ten-run variability experiment (Figure 5.19, Figure 5.20) confirms the same technique ranking at the median and at p95, with comparable separation at p99. Consistency across mean, p95, p99, and the repeated-run ranking supports practical claims about Realtime latency reductions. The principal caveat is residual tail risk: p99 estimates remain sensitive to occasional slow cloud calls, so the per-request benefit shrinks under adverse network conditions.

REST latency. The REST evidence shows smaller effect sizes: the best REST performers improve mean latency by roughly 5–15 %, compared to 35–40 % in the Realtime condition. The technique ordering is consistent with the interpretation that methods retaining more complete visual context allow the model to respond faster, while methods that aggressively crop or convert image content impose additional model-side processing cost. The REST results support the qualitative claim that compact preprocessing can reduce REST latency, with the caveat that effect magnitudes are smaller than in the Realtime condition.

Token usage and cost. Token and cost findings are reliable within a single provider/model combination. The central finding—that decoded visual geometry (crop size, image-tile count) drives cost more than compressed byte size alone—is well-supported, because the provider tiles the decoded image into fixed-size patches before counting tokens, so only methods that reduce the number of tiles produce a measurable cost reduction: `Jpeg_Q85` and `WebP_Lossy_Q85` substantially reduce serialized payload size but remain near the baseline cost, while `GlobalThumbOnly_28x28` and `Downsampler_100x100` achieve the largest cost reductions by reducing the number of billed image tiles. Cross-provider token comparisons are descriptive only, because OpenAI, Anthropic, and Google use different accounting systems; dollar cost is the safer comparison unit, but it too should be read as time-sensitive given that pricing schedules can change.

VQA-MHUG accuracy. The main accuracy findings are reliable. With 3,990 paired samples, the 99 % confidence intervals for baseline-relative accuracy change are narrow enough to distinguish tiers unambiguously. `Jpeg_Q85` and `WebP_Lossy_Q85` have confidence intervals that include zero across all tested model conditions, supporting the accuracy-neutrality claim. Saliency-only and thumbnail-only methods show losses exceeding 20 pp with intervals entirely below zero, making their degradation unambiguous. The gaze-based ROI methods occupy an intermediate tier (−3 to −11 pp depending on model capability) that constitutes a real but acceptable trade-off rather than equivalence. The principal limitation is model dependence: `Downsampler_100x100` is moderate under the Realtime and Gemini conditions but severe under GPT-5.4 (−23.5 pp), so accuracy behaviour on stronger future models cannot be inferred directly from Realtime results alone.

DriVQA and VOILA-A. Both datasets provide supporting but less definitive evidence. DriVQA contains 24 samples, so its results are strictly descriptive; its contribution is corroboration of the main pattern in a different visual domain, not an independent ranking or interval-based claim. VOILA-A is larger (500 samples) but relies on an LLM judge; reliability therefore rests on agreement between absolute rubric scores and pairwise comparisons rather than on any single number. Where the absolute-rubric and pairwise-comparison assessments agree—`Jpeg_Q85` near or above baseline, saliency-only methods substantially worse—the interpretation is reliable. Where the judge shows high instability, as for `Downsampler_100x100` at 30 % pairwise inconsistency, the absolute-score estimate is preferable to the pairwise rank.

Table 5.8 consolidates the reliability assessment by evidence area. Mapping these levels back to the research questions: the latency component of RQ1 can be answered with confidence for the Realtime condition, while the REST ranking is directional only. The accuracy component of RQ2 is reliably answered for VQA-MHUG, with the caveat that degradation estimates from the Realtime condition may understate losses on higher-capability models.

Table 5.8: Summary of reliability and principal uncertainty by result area. Realtime latency findings are the most reliable; REST latency rankings and cross-provider cost comparisons should be treated as directional only.

Result area	Reliability	Principal caveat
Realtime latency	Strong: narrow 99 % CIs, consistent across mean, p95, p99, and repeated-run diagnostics.	Absolute benefit shrinks under adverse network conditions; cloud tail risk persists for all methods.
REST latency	Moderate: direction consistent, but effect sizes small (5–15 % vs. 35–40 % in the Realtime condition).	Absolute benefit varies with model capability; techniques that reduce visual context can increase server-side processing time.
Token and cost	Reliable within provider/model: tile geometry dominates cost over compressed byte size.	Cross-provider token counts not directly comparable; pricing subject to change.
VQA-MHUG accuracy	Reliable: 3,990 paired samples with narrow CIs that clearly separate accuracy tiers.	Degree of loss is model-dependent; Realtime estimates may understate degradation on stronger models.
DriVQA	Descriptive only (n = 24); corroborates main ranking in a driving domain.	Insufficient sample size for interval-based claims or definitive ranking.
VOILA-A	Moderate: rubric and pairwise assessments consistent for the extreme-performing methods.	LLM judge instability for methods close to the baseline-quality boundary.

5.5.2. Effect Size Analysis

This subsection quantifies two complementary aspects of the findings: the run-to-run variability of the measurements themselves, and the magnitude of the observed differences in practically meaningful units. The variability results establish that the reported metrics are stable enough to support the claimed rankings; the magnitude results establish that the differences are large enough to matter in deployment.

Latency variability. The ten-run repeated experiment (10 repetitions × 50 VQA-MHUG samples using `gpt-realtime-mini`) provides a direct view of run-to-run latency stability per preprocessing technique. Figure 5.19 shows the per-sample coefficient of variation (CV) for total pipeline latency. `Grayscale` and `WebP_Lossy_Q85` are the most stable, with low median CVs and tight interquartile ranges across samples, while ROI-based methods—particularly `GazeRoi_WithGlobalThumb`—show the highest and most inconsistent variability. Notably, several preprocessors are less stable than the baseline itself, which means that faster average latency does not imply more predictable latency. Figure 5.20 complements this by expressing tail behaviour as a ratio of p95 and p99 latency against the median. `Grayscale` is the most predictable (p99 ≈ 1.5× median, small gap between p95 and p99 indicating a smooth tail), while `GazeRoi_WithGlobalThumb` produces severe p99 spikes at ≈ 3.3× median. `Jpeg_Q85` and `Baseline_NoOp` sit at ≈ 2.6× and 2.4× respectively. The large p95-to-p99 gap for these methods indicates rare but severe outliers that p95 alone would not reveal and that would exceed any fixed AR response-time budget.

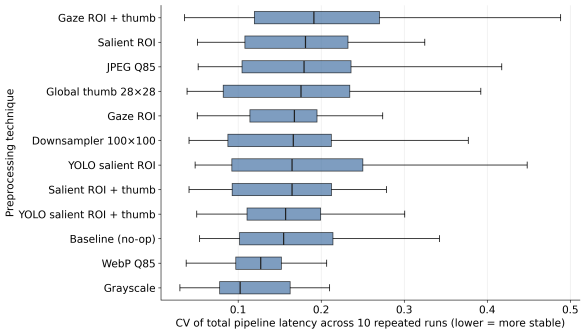


Figure 5.19: Per-sample coefficient of variation for total pipeline latency across 10 repeated Realtime runs. Lower values indicate more stable repeated latency for the same sample and preprocessing technique.

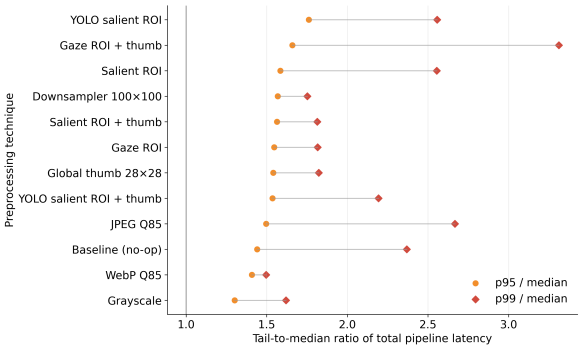


Figure 5.20: Repeated-run tail inflation ratio by preprocessing technique. The ratios compare p95 and p99 total pipeline latency with the median latency for the same technique.

Variability across pipeline dimensions. Figure 5.21 contextualises these latency findings by placing all pipeline dimensions side by side. Three tiers are visible without overlap. Payload size, input token count, and image input tokens are fully deterministic (CV = 0): these values are computed directly from the submitted image, so any run-to-run variation in other dimensions originates in network conditions, server-side inference, or LLM sampling rather than in the preprocessing pipeline itself. Pipeline latency and cached tokens show moderate variability (median CV ≈ 0.16–0.17), confirming that single-run measurements are insufficient for reliable technique ranking. Accuracy and preprocessing time are the most variable dimensions (median CV 0.49 and 0.28 respectively): run-to-run signal noise in accuracy substantially exceeds that in latency, which is the key motivation for the large-sample confidence-interval design used throughout the main experiments.

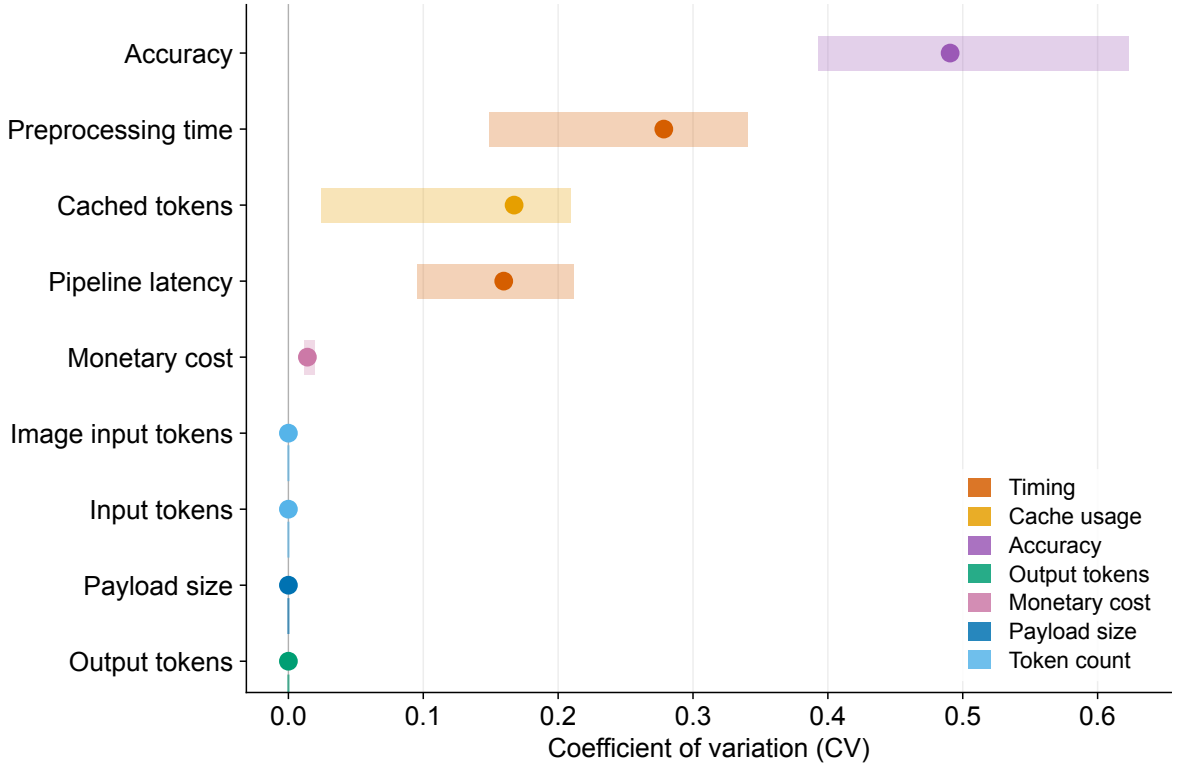


Figure 5.21: Coefficient of variation across all pipeline dimensions for the 10 × 50 repeated-run experiment. Dimensions with CV = 0 (payload size, token counts) are fully deterministic; pipeline latency and accuracy show meaningful run-to-run variability originating in network and server-side conditions rather than in the preprocessing pipeline itself.

Accuracy non-determinism. Figure 5.22 reveals the structure of accuracy non-determinism at the per-sample level. Each row shows the fraction of the 50 repeated samples answered correctly in exactly k out of 10 runs. The distribution is strongly bimodal: most samples are either consistently correct (column 10) or consistently wrong (column 0), with the inconsistent middle columns (1–9) accounting for approximately 38 % of sample–preprocessor pairs. This means that single-run accuracy is a noisy estimator for a substantial fraction of samples, but the noise is concentrated in borderline cases rather than spread uniformly. Critically, the always-correct fraction ranges from 18 % for `GlobalThumbOnly_28x28` to 56 % for `WebP_Lossy_Q85`—a 38 pp spread that reflects a reliability gap rather than purely a mean-accuracy gap. `WebP_Lossy_Q85` and `GazeRoi_NoGlobalThumb` combine high always-correct rates (56 % and 52 %) with low inconsistency (28 % and 30 %), confirming that their single-run accuracy estimates are genuine signals rather than artefacts of lucky evaluation runs. By contrast, `GlobalThumbOnly_28x28` has the highest inconsistency rate (52 % of samples in columns 1–9): it not only loses accuracy but does so unpredictably.

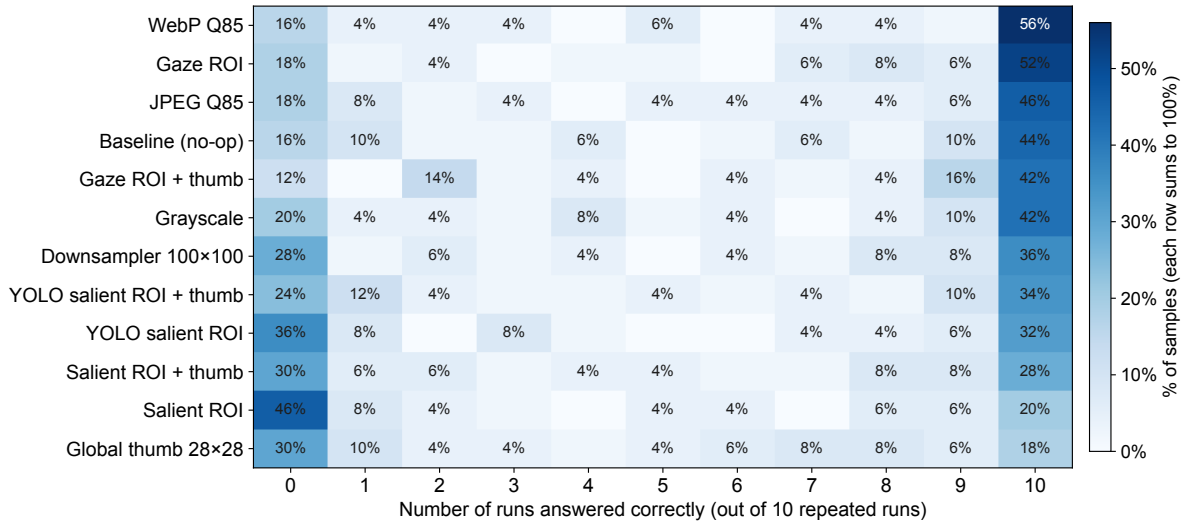


Figure 5.22: Per-sample correctness consistency heatmap across 10 repeated Realtime runs. Each cell shows the percentage of the 50 test samples answered correctly in exactly k out of 10 runs; every row sums to 100 %. Columns at 0 and 10 represent consistently wrong and consistently correct samples respectively; columns 1–9 indicate non-deterministic answers whose single-run score depends on run order. The strongly bimodal distribution confirms that single-run accuracy is reliable only for the consistently-correct and consistently-wrong fractions; the 38 % borderline cases require multi-run aggregation to distinguish genuine effects.

Efficiency effect sizes. Figure 5.23 quantifies baseline-relative changes in the units most relevant to deployment, using the OpenAI Realtime Mini condition as the primary reference. Positive values indicate reductions relative to `Baseline_NoOp`; negative values indicate regressions. All figures use this condition because token and cost accounting differ across providers. The central finding is that reducing serialized payload and reducing billed input usage are not the same result. `Jpeg_Q85` and `WebP_Lossy_Q85` achieve substantial payload reductions—roughly 50–60 %—but their token and cost bars remain near zero, because the provider tiles the decoded image into fixed-size patches and charges for the decoded visual geometry rather than the compressed byte stream. Optimising for network transfer and optimising for API cost therefore require different preprocessing strategies.

Methods that change the submitted visual geometry avoid this limitation. `GlobalThumbOnly_28x28`, `GazeRoi_NoGlobalThumb`, and `Downsampler_100x100` achieve 30 %+ reductions in both latency and estimated cost by reducing the number of billed image tiles. By contrast, methods that submit both a global thumbnail and a ROI crop—such as `GazeRoi_WithGlobalThumb`—can register negative cost effects: even when each image is individually compact, the two-image submission may cross the provider's single-image tile boundary and increase the billed tile count.

Together, these efficiency and accuracy measurements provide the empirical basis for the deployment guidance, failure-mode analysis, and comparison with prior work in Chapter 6.

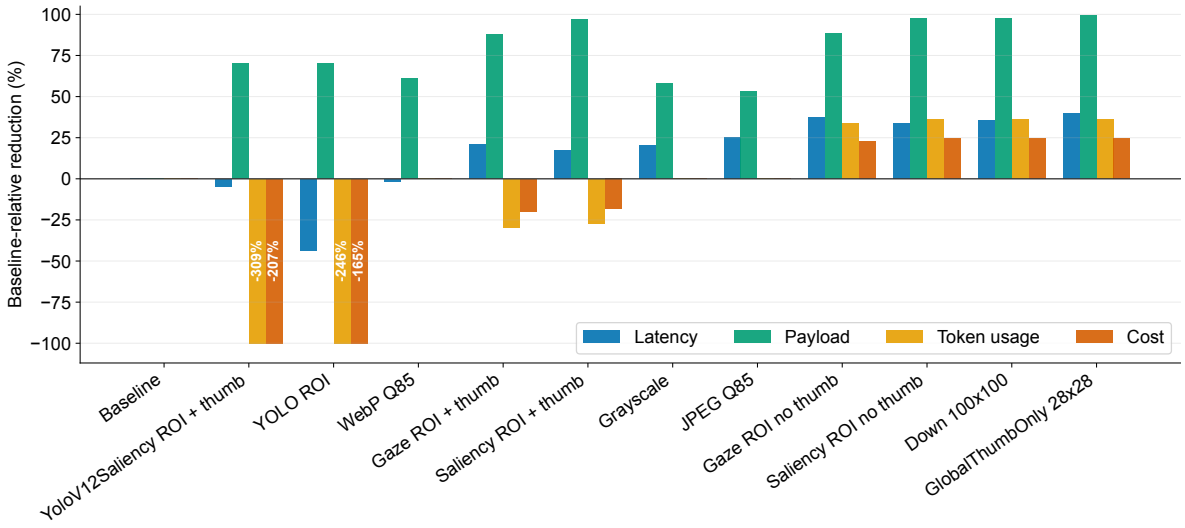


Figure 5.23: Baseline-relative efficiency effect sizes by preprocessing technique for the OpenAI Realtime Mini condition. Positive values indicate reductions relative to `Baseline_NoOp`; negative values indicate increases or regressions. Payload reduction and cost reduction are not equivalent, because the provider bills for decoded image tile count rather than compressed byte size.

6

Discussion

This chapter interprets the empirical results from Chapter 5 along three lines: the efficiency–accuracy trade-off landscape that shapes practical deployment choices, scenario-specific technique recommendations and their known failure modes, and a comparison of the findings with the prior work surveyed in Chapter 3.

6.1. Efficiency–Accuracy Trade-off Landscape

The primary trade-off for AR use is between total pipeline latency and answer quality. It must be noted that this analysis will be conducted on the Realtime model since it is the most appropriate model for Mobile AR devices. Figure 6.1 plots a Pareto frontier comparing total pipeline latency of the Realtime-mini model on the x-axis and VQA-MHUG accuracy on the y-axis. Lower latency and higher accuracy are preferable, so the upper-left region represents the best practical trade-off. The baseline is included as the reference point, and the Pareto frontier identifies methods that are not dominated by another method on both mean latency and mean accuracy. The frontier should therefore be read as a descriptive decision aid; uncertainty and practical reliability are interpreted from the confidence-interval, paired-difference, and tail-latency analyses in section 5.5.

The frontier points are `GlobalThumbOnly_28x28`, `GazeRoi_NoGlobalThumb`, `Jpeg_Q85`, and the no-preprocessing baseline. These points define the measured Realtime trade-off: the thumbnail-only method is the fastest but loses too much accuracy (−24.5 pp), the gaze-only ROI method gives a large latency reduction (38%) with a moderate accuracy loss of 3%, JPEG compression gives the strongest near-baseline accuracy point while still reducing latency substantially by 25%, and `Baseline_NoOp` remains the reference for maximum unprocessed visual context. For an AR device with reliable gaze tracking, `GazeRoi_NoGlobalThumb` is therefore the preferred AR-native point on this frontier: it uses a sensor signal already available in many AR settings and converts it into larger latency and payload gains than JPEG. It should, however, be described as a practical trade-off rather than as accuracy-neutral. When gaze is not available or when preserving answer quality is the dominant requirement, `Jpeg_Q85` is the safer general default because its accuracy remains closest to the baseline. subsection 5.2.3 shows that the choice of JPEG quality should be determined based on the image input format. DriVQA showed an increase in payload size due to the original encoding format being of less quality than JPEG Q85.

The second trade-off for AR use is between total cost in USD and Accuracy. Cost matters because a real AR system may issue many multimodal requests during prolonged use. Figure 6.2 compares estimated cost per 1,000 requests of the Realtime-mini model with VQA-MHUG accuracy. This view produces a slightly different ranking from the latency–accuracy plot.

In Figure 6.2, `GazeRoi_NoGlobalThumb` is the most useful pareto frontier point for different reasons: gaze ROI, reaches the low-cost region with a reduction of 20%, has a large latency reduction (38%) but a small

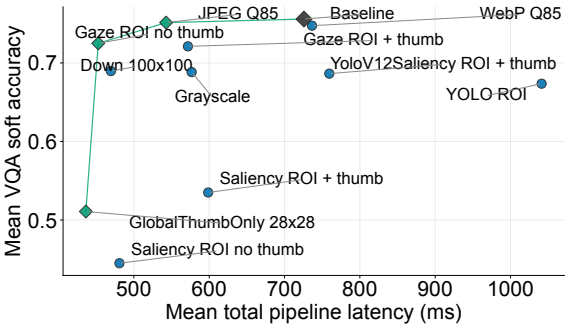


Figure 6.1: Latency–accuracy trade-off by preprocessing technique for Realtime-mini on the VQA-MHUG dataset. GazeROI is the Pareto-dominant choice when latency reduction takes priority; JPEG Q85 is Pareto-dominant when accuracy preservation is required.

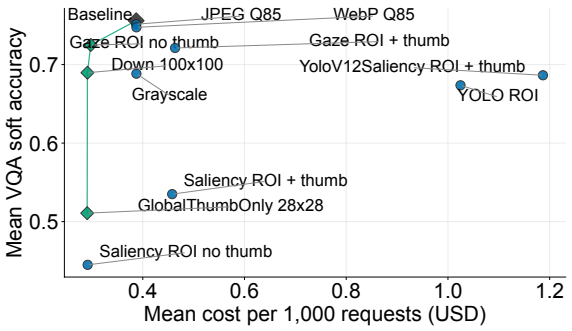


Figure 6.2: Pareto cost–accuracy trade-off by preprocessing technique for the Realtime model on the VQA-MHUG dataset. GazeROI is the only non-dominated technique; it reaches the low-cost region with a small accuracy penalty.

accuracy penalty (3%). For an AR deployment where gaze is already part of the sensing stack, this makes gaze ROI the stronger cost-aware AR strategy; for deployments without dependable gaze, baseline remains the more robust quality-preserving choice with regard to cost. `Downsampler_100x100`, `GlobalThumbOnly_28x28`, and `SalientRoi_NoGlobalThumb` are cheap, but their accuracy losses make them less suitable as default AR choices.

6.2. Deployment Recommendations

6.2.1. Scenario-Specific Technique Selection

This section translates the measured trade-offs into scenario-specific deployment guidance using a weighted multi-criteria analysis. The Pareto frontier plots (Figure 6.1, Figure 6.2) are useful for two-metric interpretation, but deployment choices usually depend on more than two metrics at once. Figure 6.3 therefore summarizes a weighted decision analysis for the OpenAI Realtime Mini condition. Each column represents a practical decision scenario, and each cell reports a normalized weighted score together with its rank within that scenario. The matrix should be read as a structured sensitivity analysis rather than a universal utility function: changing the weights changes the score, but the ranking reveals which methods remain competitive across plausible AR priorities.

The scenario scores are computed from normalized accuracy, latency, and cost scores, where larger values are better. For each provider/model, the three raw metrics were min–max normalized across the same set of preprocessing techniques. Accuracy was scaled as $s = (x - \min(x))/(\max(x) - \min(x))$, while latency and cost were inverted after min–max scaling, $s = 1 - (x - \min(x))/(\max(x) - \min(x))$, so that larger normalized values always indicate better performance. No baseline-relative scaling or clipping was applied; the best observed technique for each metric receives a score of 1, the worst receives 0, and all weighted decision scores are computed from these provider-specific normalized values.

The seven scenarios span the range of plausible AR operator priorities, from latency-critical real-time assistants to accuracy-critical inspection tools (Table 6.1). The scenario weights are illustrative rather than empirically grounded; no user study was conducted to elicit practitioner preferences. The matrix should therefore be read as a structured sensitivity analysis — the key question is not which scenario scores highest, but whether a technique's rank remains stable across scenarios with different priorities.

The matrix strengthens the AR-specific recommendation. `GazeRoi_NoGlobalThumb` ranks first in the balanced, latency-first, accuracy-first, cost-first, Realtime-quality, and budget-Realtime scenarios, and second only in the quality-on-budget scenario. GazeROI's first-place rank is therefore not metric-specific for the Realtime model: it leads whether the weighting favours latency, cost, accuracy, or a combination. `Jpeg_Q85` becomes the best method only when the scenario places the strongest emphasis on quality under a budget constraint, and it remains near the top in accuracy-oriented scenarios. `Downsampler_100x100` is consis-

Table 6.1: Weights assigned to accuracy, latency, and cost for each of the seven decision scenarios used in the weighted trade-off analysis.

Scenario	Accuracy	Latency	Cost
Balanced	0.333	0.333	0.333
Latency first	0.25	0.50	0.25
Accuracy first	0.50	0.25	0.25
Cost first	0.25	0.25	0.50
Realtime quality	0.40	0.40	0.20
Budget realtime	0.20	0.40	0.40
Quality on budget	0.60	0.10	0.30

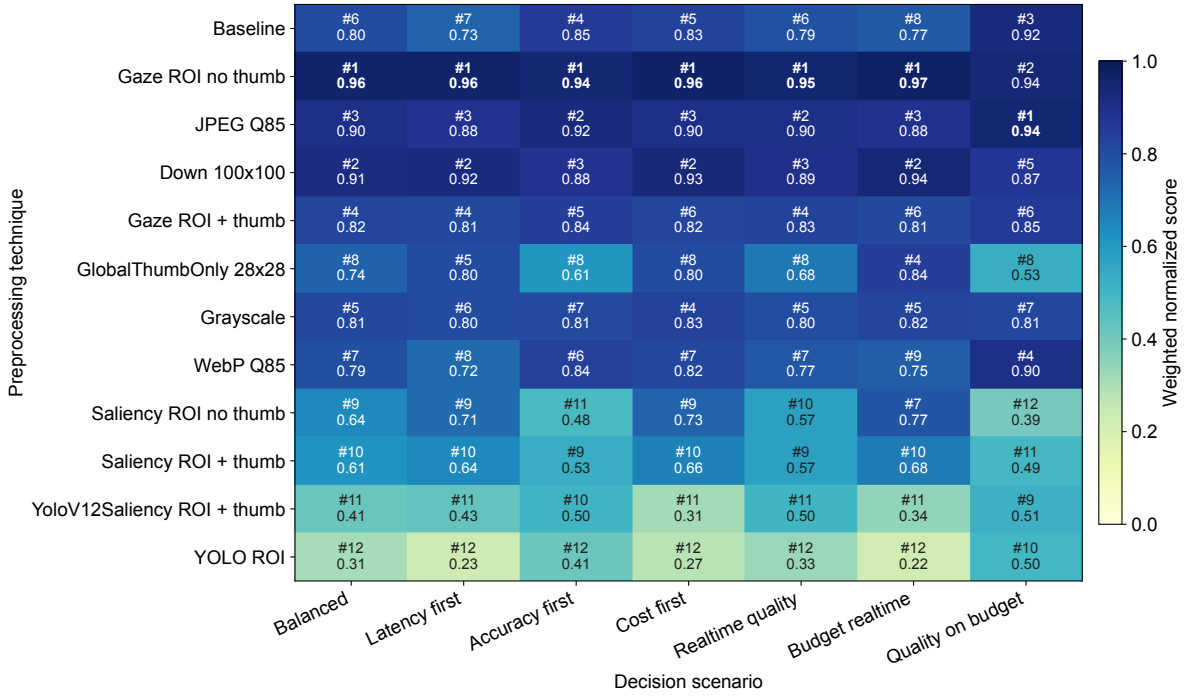


Figure 6.3: Weighted trade-off decision matrix by preprocessing technique for the OpenAI Realtime Mini condition. Each cell reports the scenario rank and normalized weighted score; higher scores are better. GazeROI ranks first in six of seven scenarios, confirming that its recommendation is not driven by any single metric.

tently competitive for latency and cost, but its larger accuracy loss prevents it from becoming the preferred balanced AR method. `GlobalThumbOnly_28x28` appears attractive in budget-oriented scoring because it is extremely efficient, but its poor accuracy rank confirms that it is better interpreted as an efficiency lower bound than as a general recommendation.

Figure 6.4 tests the same conclusion across a broader sweep of accuracy–latency–cost priorities. Rather than relying on one fixed scenario, it counts how often each method wins across all 66 weight combinations — a direct test of whether the GazeROI recommendation depends on a particular weighting choice. The 66 combinations are obtained by enumerating every integer split of ten discrete units across the three objectives (accuracy, latency, cost), so that the weights sum to 1.0 in steps of 0.1. This gives the triangular number $T(11) = 66$ points on the three-dimensional simplex. For each combination, each preprocessor receives a weighted sum of its min–max normalised scores (accuracy: higher is better; latency and cost: lower is better), and the preprocessor with the highest score is recorded as the winner. Counting these wins across all 66 combinations yields the share reported in the figure.

The winner-frequency analysis shows that `GazeRoi_NoGlobalThumb` is the most robust overall trade-off for the Realtime model, winning 64% of the tested weight combinations. `GlobalThumbOnly_28x28` wins 17%, reflecting extreme efficiency-oriented preferences, while `Jpeg_Q85` wins 12%, corresponding to quality-preserving preferences. The no-preprocessing baseline wins 8%, which is expected when accuracy is

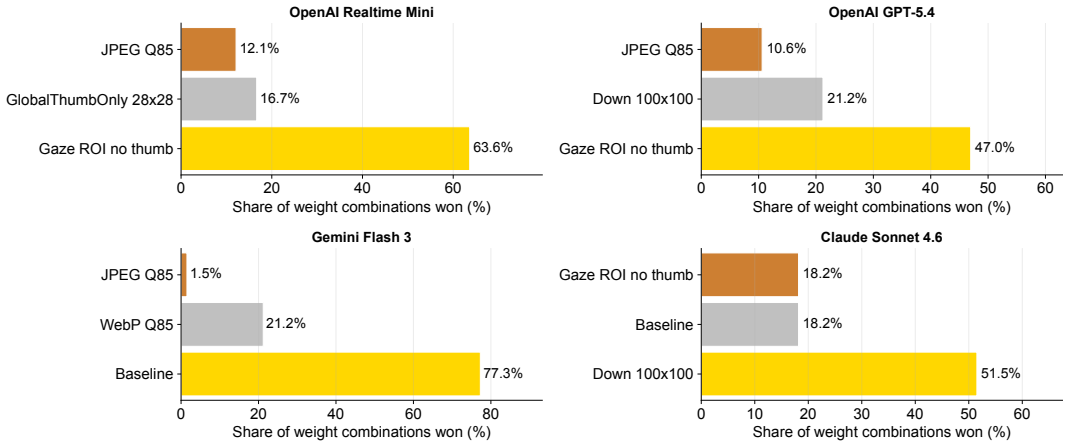


Figure 6.4: Winner frequency across 66 accuracy–latency–cost weight combinations for the OpenAI Realtime Mini condition. GazeROI wins 64 % of tested combinations, confirming it as the robust AR-native recommendation; JPEG Q85 wins 12 %, corresponding to quality-preserving weight preferences.

weighted far above efficiency. No other method wins any of the tested combinations. The winner-frequency distribution supports a two-method recommendation: `GazeRoi_NoGlobalThumb` for AR deployments with reliable gaze tracking, and `Jpeg_Q85` as the accuracy-preserving fallback when gaze is unavailable or unreliable.

The cross-provider matrices differ substantially from the Realtime Mini result and are collected in section A.6 for reference. Figure 6.4 shows the top three results for each model for the winner-frequency analysis. For `gpt-5.4`, `GazeRoi_NoGlobalThumb` ranks first across all seven scenarios, consistent with gaze ROI's large payload and cost reductions on a high-cost model tier where per-token pricing makes those savings more valuable. For `claude-sonnet-4-6`, `Downsampler_100x100` wins six of seven scenarios; in the REST condition, gaze ROI's server-side latency benefit is modest and the downsampler reaches comparable efficiency at a lower normalized accuracy cost, though `GazeRoi_NoGlobalThumb` recovers to first in the accuracy-first scenario. The `gemini-3-flash-preview` matrix is the most divergent from Realtime Mini: the no-preprocessing baseline ranks first in six of seven scenarios, since most preprocessing techniques added latency on that provider rather than reducing it. These matrices are a cross-provider sensitivity check, not primary deployment guidance; Claude Sonnet and Gemini Flash 3 were each evaluated on 500 samples, so the Realtime Mini matrix and winner-frequency analysis remain the basis for the main recommendations.

`Jpeg_Q85` also has an energy advantage on embedded AR hardware, beyond the latency and payload reductions already measured. Wireless image transmission is a dominant power drain on wearable devices [53], and a roughly 50 % payload reduction lowers that cost by the same proportion. Q85 and Q100 share the same DCT algorithm and differ only in the quantization table applied to each 8×8 block [108], so the extra encoding work on the device is negligible compared to what is saved in transmission. On devices whose image sensor already produces JPEG output, the Q85 quality level can be set at the sensor, removing the preprocessing step from the AR pipeline entirely.

Taking all of the above together, the deployment recommendation follows a clear decision rule. When reliable gaze tracking is available, use `GazeRoi_NoGlobalThumb`: it reduces total pipeline latency by 38 %, cuts cost by 20 %, and pays only a ≈3 percentage-point accuracy penalty on Realtime Mini. When gaze is unavailable or unreliable, use `Jpeg_Q85`: it reduces latency by 25 % and payload by roughly 50 % while keeping VQA-MHUG accuracy within 1 percentage point of the unprocessed baseline, and possibly reduces energy usage. All other tested techniques are dominated on the Realtime Mini Pareto frontier or introduce latency regressions, and are not recommended as defaults.

The recommendations above reflect average-case behaviour under normal operating conditions. The following subsection documents the systematic failure modes and scope limitations that bound their applicability.

6.2.2. Failure Cases and Known Limitations

Technique-Level Failures

Several techniques fail in predictable, deployment-relevant ways. Both YOLOv12 variants increase total pipeline latency (+5% and +43%) because local ONNX inference takes 87–94 ms on CPU, exceeding any server-side savings. The 290 ms server-side gap between the two YOLO variants likely reflects the additional image submitted with the thumbnail variant, which increases the model's processing burden. `WebP_Lossy_Q85` fails from excessive client-side encoding cost and additional server-side work: despite producing the smallest serialized file, its encoding cost (~91 ms, the highest of all methods) is compounded by the fact that providers likely decode and re-encode the WebP image internally, negating the file-size savings. `Jpeg_Q85` produces a payload anomaly on DriVQA: those images are already compressed at lower quality than Q85, so re-encoding raises the file size. This is a practical hazard for embedded AR devices with already-compressed sensor outputs.

Two further methods trade away accuracy so severely that they fall outside the useful operating range. `SalientROI_NoGlobalThumb` is the worst technique for accuracy across every model tested (−31 pp on Realtime Mini, −35 pp on GPT-5.4). On VOILA-A it produces a 39 % overall quality drop and the highest hallucination rate of any method (unsupported-claim score 0.584 vs. 0.326 baseline), indicating that saliency-guided crops frequently discard the answer region entirely, as the saliency detector responds to texture and contrast rather than semantically question-relevant objects. The method also regresses on REST latency (+22–25%), whereas it reduced Realtime latency. This contrast is consistent with the REST model requiring more visual context to process a request quickly: removing the salient region forces the model to reason from an ambiguous crop, which increases server-side response time. `GlobalThumbOnly_28x28` is the fastest technique (−40 % latency) but the second worst for accuracy (−24.5 pp on Realtime, −41.5 pp on GPT-5.4) and exhibits the highest non-determinism rate (52 % of samples vary across ten repeated runs), making single-run accuracy estimates for this method unreliable.

`GazeRoi_NoGlobalThumb` hides a model-tier dependency. Its accuracy loss is acceptable on Realtime Mini (−3 pp) but substantially larger on stronger models (−10 pp on GPT-5.4, −8.5 pp on Claude Sonnet). Higher-capability models can answer questions whose relevant region falls outside the gaze crop; the lower-accuracy Realtime model fails those samples regardless, masking the degradation in Realtime-only evaluations. GazeROI also shows the highest latency variability on DriVQA, with a 2250 ms p99 outlier, because driving questions (e.g., "how many vehicles are in the frame?") require global scene context that a single fixation-based crop cannot provide. A separate configuration constraint compounds this: the gaze attention heatmap was computed at a fixed `roi_detect_size` of 64 × 64 pixels for all experiments. This resolution is coarse relative to high-resolution inputs and cannot be tuned per question or scene type. A dynamically adjustable detection resolution — finer for detail-sensitive questions, coarser for global-scene tasks — could reduce both the missed-region failures and the latency tail without changing the cropping mechanism itself. `Downsampler_100x100` exhibits an analogous model dependency: accuracy loss is moderate on Realtime (−6.6 pp) and Gemini Flash (−2.5 pp) but severe on GPT-5.4 (−23.5 pp), where fine-grained spatial detail that a 100×100 downsample destroys becomes decision-critical.

Systematic Measurement Limitations

The largest source of latency variance in this study is not preprocessing. The baseline latency gap between the fastest provider (gpt-realtime-mini, 784 ms) and the slowest (gemini-flash, 1796 ms) is ~1012 ms, which is larger than the entire preprocessing technique spread (~509 ms). Preprocessing is therefore a secondary latency lever; model and API selection is the primary one.

Compression methods do not reduce token usage or billed cost. JPEG Q85 and WebP Q85 reduce serialized payload by 50–60 % but leave token counts and estimated cost near baseline, because providers tokenize the decoded pixel grid rather than the compressed byte stream. Developers targeting cost reduction must use geometry-changing methods (crops, downsampling, or thumbnails) rather than compression alone. The same limitation applies to latency for REST providers: the JPEG-quality calibration reported in subsection 5.2.3 finds $R^2 \leq 0.02$ between serialized byte size and total pipeline latency for the three REST interfaces, confirming that server-side compute, not transmission time, governs REST response time. Adding

a global thumbnail to GazeROI creates a two-image submission that can cross provider tile boundaries and increase billed tile count, negating the per-crop cost savings.

One instrumentation artefact affects preprocessing-time estimates. For cheap operations (grayscale, downsampler, gaze ROI), the Realtime WebSocket pipeline reports 2–3× higher preprocessing times than REST for the same operations. This is system-level scheduling jitter, not real preprocessing work, and it makes per-sample overhead measurements for these techniques unreliable under the WebSocket protocol.

Scope and Ecological Limitations

All datasets used in this thesis present questions as pre-typed text strings. This was not a free design choice. Gaze-annotated VQA datasets that simultaneously provide eye-tracking data and voice-format queries do not exist; the three datasets selected (VQA-MHUG, VOILA-A, and DriVQA) were chosen because they carry gaze annotations, and all three use typed questions. In a deployed AR assistant, however, voice is the natural interaction mode: the user speaks a query, a speech-to-text model transcribes it, and the transcription is forwarded to the MLLM alongside the preprocessed image. This thesis measures none of those steps. The automatic speech recognition (ASR) stage introduces its own latency — Cloud ASR services typically report real-time transcription latencies in the range of 200–600 ms depending on query length, provider, and streaming configuration, with mainstream providers such as Google Cloud Speech-to-Text reporting 300–600 ms under standard conditions [102, 9, 44]. — and this cost is absent from the `total_pipeline_ms` figures reported here. A deployed voice-driven AR pipeline would therefore have a higher end-to-end latency baseline than reported, with an additional component that preprocessing cannot reduce. Beyond latency, voice queries differ structurally from typed VQA questions: they are shorter, less formally phrased, and more often context-dependent. The benchmark datasets do not reflect this query distribution, so the accuracy results may not transfer directly to a voice-driven deployment.

The study measures a controlled offline pipeline, not a deployed AR system. It uses static images and pre-sampled gaze traces, and does not capture perceived responsiveness, wearable compute constraints, battery drain, thermal throttling, camera capture latency, gaze calibration lag, or display rendering time. The reported `total_pipeline_ms` is a reproducible per-sample metric but not user-facing end-to-end latency.

All experiments ran on a single MSI laptop (Intel i7-8750H, 16 GB RAM). Local preprocessing times are hardware-specific; results on embedded AR hardware, which typically has lower CPU performance and stricter power envelopes, would differ.

The results are also temporally bounded. All measurements were collected during April–May 2026 under specific API configurations. Model behavior, token accounting, pricing tiers, and serving infrastructure are subject to change, and the quantitative rankings may not hold for future model versions or provider updates.

Two dataset-level limitations affect confidence in specific sub-results. VOILA-A contains many vague or outdated reference answers (209 of 709 candidates were discarded), and current models regularly produce answers that exceed the reference quality. LLM judge instability is elevated for at least one technique (30 % for `Downsampler_100x100`), which widens the effective uncertainty around those accuracy estimates. DriVQA comprises only 24 samples, so all DriVQA results are strictly descriptive; the baseline model accuracy of 52 % further limits the diagnostic value of preprocessing comparisons on that dataset.

Three further limitations narrow the scope of gaze-based and cross-provider conclusions. The study uses VOILA-COCO with mouse-trace proxies because the gaze-tracked VOILA-GAZE subset was not publicly released; the proxy may not accurately represent real fixation behaviour. For samples with multiple participant recordings, one trace was selected at random (seed 12345), so gaze-based results reflect a single recording per sample rather than an average over participant behaviour. Cross-provider coverage is limited: Anthropic Claude Sonnet and Gemini Flash were evaluated on only 500 VQA-MHUG samples and serve as robustness checks rather than primary benchmarks, so the main comparative conclusions are OpenAI-centric.

Of these limitations, the most consequential for the primary recommendations are the model-tier dependency of GazeROI (which means the −3 pp Realtime result cannot be projected to stronger models without re-evaluation). The dataset annotation quality, hardware specificity, and temporal validity issues are real but secondary: they affect the precision of individual numbers, not the direction of the core finding that GazeROI

and JPEG Q85 form the practical Pareto frontier under the Realtime Mini condition.

Having established what the findings are and where they break down, the following section places them in the context of the prior work reviewed in Chapter 3.

6.3. Comparison with Related Work

6.3.1. Gaze-guided ROI Preprocessing

Gaze-guided two-scale preprocessing — a high-resolution ROI crop combined with a low-resolution 28×28 global thumbnail — was established on open-weight VLMs by Chen and Qi [18], achieving token reductions of up to 93% relative to the full-image baseline while maintaining task accuracy. Within their comparison, the two-scale variant (ROI plus thumbnail) consistently outperformed ROI-only: for the 3B model it produced a win-rate of approximately 51–57% over ROI-only across gaze-mass thresholds, with most of the gain coming from cases where the crop alone missed global context cues such as object counts or spatial layout. For the 7B model the advantage was smaller but still consistent (50–53%). Both variants remained well within the efficiency advantage over the full-image baseline; the thumbnail added only modest overhead relative to the large saving from the crop itself.

This thesis implements the same gaze-guided principle and the same two structural variants, but deploys them on commercial cloud APIs rather than local open-weight models. Comparing commercial cloud deployment to the open-weight setting of Chen and Qi [18] produces a systematic reversal of the within-variant ordering. On Realtime Mini and Claude Sonnet, `GazeRoi_NoGlobalThumb` achieves better accuracy preservation than `GazeRoi_WithGlobalThumb` (-3.1 pp vs. -3.5 pp on Realtime Mini; -8.5 pp vs. -9.4 pp on Claude Sonnet) and a substantially larger latency reduction (38% vs. 21% on Realtime Mini). The reversal is not uniform across all model conditions: on GPT-5.4, the thumbnail variant is marginally better (-9.5 pp vs. -10.2 pp), and on Gemini Flash 3, the thumbnail variant is substantially better (-2.2 pp vs. -4.9 pp). Gemini Flash 3 applies flat-rate 258-token accounting for images below the 384-pixel threshold [37], meaning the thumbnail stays within the same accounting tier and avoids the tile-boundary penalty seen on OpenAI; GPT-5.4 exhibits the same directional tension but the accuracy difference is small. The dominant pattern across the four conditions — and the strongest one for the latency-sensitive AR use case on Realtime Mini — is that `GazeRoi_NoGlobalThumb` is the more efficient choice on commercial APIs with tile-based accounting.

The token-accounting mechanism explains this pattern. On open-weight VLMs, accounting is linear and model-internal: adding a small thumbnail costs a proportionally small token increment that is easily offset by the context it provides. On commercial cloud VLMs with tile-based accounting, the same thumbnail may push the total submission above a tile boundary, adding a disproportionate token and cost penalty. Methods that submit a single compact image — `GazeRoi_NoGlobalThumb`, `Downsampler_100x100`, `GlobalThumbOnly_28x28` — all fall in the low-cost region; adding a thumbnail converts an efficient single-image request into a two-image submission that can exceed the provider's tile threshold and negate the per-crop savings [77]. Neither variant is universally better; the right choice depends on the provider's tokenization scheme. Practitioners migrating a gaze-guided preprocessing pipeline from open-weight to commercial APIs should verify the provider's per-image tile threshold before choosing between ROI-only and ROI-plus-thumbnail.

Two implementation differences between this thesis and Chen and Qi [18] may further affect the comparison. First, their implementation uses dynamic ROI scaling, adapting the detection resolution to the input image; this thesis uses a fixed ROI detection size of 64×64 pixels for all experiments. When fixations are distributed across a scene, a fixed-size crop loses global context that dynamic scaling would retain by expanding the window. This makes the global thumbnail more valuable under fixed-size cropping than under dynamic scaling: with a fixed 64×64 ROI, `GazeRoi_WithGlobalThumb` can compensate for the missing global context through the thumbnail, whereas `GazeRoi_NoGlobalThumb` cannot. Under dynamic scaling, the ROI itself already captures that global context, reducing the marginal value of the thumbnail. The ordering difference between `GazeRoi_WithGlobalThumb` and `GazeRoi_NoGlobalThumb` should therefore be larger under a fixed-size crop than under dynamic scaling. Conversely, introducing dynamic ROI scaling would be expected to narrow this gap, as the thumbnail's compensatory role diminishes once the local

crop already carries the global context. This narrowing only applies when fixations are spread enough for the dynamic ROI to exceed the 64 × 64 minimum; when gaze is concentrated, the minimum size constraint within dynamic scaling floors the crop at 64 × 64, producing the same window as the fixed baseline, and the thumbnail retains its full compensatory value. For the VQA-MHUG and VOILA-A datasets, whose images are approximately 480 × 360 pixels, the fixed size is a reasonable approximation, but for the driving images in DriVQA — which are substantially larger — the fixed detection resolution is coarser relative to the scene scale. The high latency variability and 2 250 ms p99 outlier observed for the gaze ROI method on DriVQA (Figure 5.4) are consistent with this limitation: driving questions frequently require global scene context (e.g., vehicle count, spatial layout) that a coarser fixed-size crop is unable to provide [22]. A dynamically adjusted detection resolution, finer for detail-sensitive questions and coarser when broader coverage is needed, might reduce both the missed-region failures and the latency tail on high-resolution inputs without modifying the cropping mechanism. Second, Chen and Qi [18] mention a minimum ROI size constraint but do not specify its value; this thesis enforces a minimum ROI size of 64 pixels based on this guidance. Across the reported conditions, this threshold appears adequate.

6.3.2. Region-Selective Methods: Saliency and Object Detection

The related work observes that task-independent saliency maps correlate poorly with the image regions humans attend to when answering VQA questions [27, 112, 29], and that object-detection crops may return redundant or irrelevant regions in crowded scenes [3, 29]. The experimental results support both characterizations. `SalientRoi_NoGlobalThumb` is the weakest technique for accuracy across every model and dataset, producing accuracy losses of 31–36 pp on VQA-MHUG together with the highest hallucination rate on VOILA-A, consistent with the saliency detector responding to texture and contrast rather than the semantically question-relevant region. The YOLOv12-based variants increase total pipeline latency by 5–43% relative to baseline, replicating the finding that local DNN-based detection overhead dominates any server-side saving on constrained hardware [3]. The REST condition, where the model requires more visual context to produce a fast response, shows saliency methods increasing latency by 22–25% — an additional confirmation that removing globally relevant content forces the model to work harder rather than less.

6.3.3. Compression and Global Reduction Methods

The related work identifies JPEG [108] and WebP [32] as deployment-ready payload-reduction methods, and notes that aggressive downscaling can degrade recognition by stripping high-frequency detail [104]. Both findings are replicated here. `Jpeg_Q85` and `WebP_Lossy_Q85` are the only two preprocessing techniques whose 99% confidence intervals include zero across all four model conditions, confirming that moderate lossy compression does not produce a measurable accuracy penalty at Q85. At the same time, `Downsampler_100x100` shows a model-dependent loss that is moderate under Realtime Mini (−6.6 pp) but severe under GPT-5.4 (−23.5 pp), consistent with the finding of Res-Bench [56] that accuracy degradation under resolution reduction is strongly model-dependent: fine spatial detail, once destroyed by downsampling, becomes decision-critical when the model is capable of exploiting it. Grayscale conversion produces consistent losses of −5 to −9 pp, in line with the evidence from ColorBench that chromatic information contributes to visual reasoning on color-dependent tasks [58].

Among the deployment contexts identified in the related work, `Jpeg_Q85` is the technique most directly applicable to accessibility-focused visual assistants for blind and low-vision (BLV) users. Deployed services such as Be My Eyes [10] rely on commercial cloud VLMs where accuracy cannot be traded away and each request carries both a latency cost and a monetary cost that affect the user experience during real-time navigation or scene interpretation. VLM studies for BLV navigation document that accuracy failures are consequential in these high-stakes contexts [68]: a mis-described intersection or misidentified obstacle is not a statistical abstraction but a practical hazard. The accuracy-neutral property of `Jpeg_Q85` is therefore a prerequisite rather than a preference for this use case, since region-selective techniques that introduce even a small accuracy penalty are unsuitable when correctness is the dominant constraint. At the same time, the ≈25% latency reduction and ≈50% payload reduction achievable with `Jpeg_Q85` reduce both cost per query and perceived response time — both practically relevant when a user is moving through a real environment and waiting for a response. Failure modes specific to BLV-captured images, such as blur and poor framing documented in

long-form and consistency-aware VQA studies [42, 19], may also interact with the compression step; Q85 operates on whatever pixel content the sensor captures, so these pre-existing quality issues are not introduced or amplified by the compression itself.

6.3.4. Contextualizing the Main Contribution

Taken together, these comparisons show that the qualitative findings from the open-weight and local-inference literature do transfer to the commercial cloud setting — with one important exception. The relative merit of ROI-only versus ROI-plus-thumbnail preprocessing inverts on providers with tile-based accounting when moving from open-weight to commercial deployment. This inversion is not a contradiction of prior work; it is a boundary condition that prior work, which did not target commercial APIs, was not positioned to identify. The evidence from this thesis therefore complements and extends the preprocessing literature rather than overturning it: it establishes where open-weight findings generalize and where the commercial deployment surface imposes constraints that change the optimal design choice.

A further gap becomes visible when this thesis is placed alongside the wearable and smart-glass benchmarks reviewed in section 3.4. WearVQA [15], SUPERGLASSES [47], and CRAG-MM [109] each evaluate a broad model set that spans both open-source local-inference models and commercial REST endpoints, enabling direct cross-paradigm comparison within a single benchmark. This thesis evaluates only commercial cloud APIs (REST and Realtime WebSocket); local-inference models are entirely absent from the experimental design.

7

Conclusion

This chapter closes the thesis. The first section answers the three research questions directly, drawing on the results and deployment analysis from Chapters 5 and 6. The second section states the methodological, empirical, and practical contributions. The third section identifies directions for future work.

7.1. Answers to the Research Questions

7.1.1. RQ1: Efficiency Impact of Preprocessing

End-to-end latency in the pipeline is governed by three factors, operating in decreasing order of influence. The first and dominant factor is provider and model selection. On the shared 500-sample VQA-MHUG subset, mean baseline latency ranges from 784 ms for `gpt-realtime-mini` to 1,796 ms for `gemini-3-flash-preview`, a spread of 1,012 ms that exceeds the 509 ms span across all tested preprocessing techniques on the Realtime model. No preprocessing method closes this gap: even the most compact representation applied to a slower provider cannot reach the unprocessed baseline latency of a faster one. Provider selection is therefore a prerequisite before preprocessing gains become relevant.

The second factor is task formulation. The same model produces different baseline latencies across datasets: 615 ms for VOILA-A, 726 ms for VQA-MHUG, and 863 ms for DriVQA, all under `gpt-realtime-mini`. These differences reflect visual domain and question complexity rather than image content alone. Prompt length and question structure affect how long the model takes to generate a response, independently of any preprocessing applied to the image.

The third factor is preprocessing technique, and within this factor the gains are practically meaningful. In the Realtime condition, the three strongest techniques reduce total pipeline latency by 35–40%: `GlobalThumbOnly_28x28` reaches 436 ms (−40%), `GazeRoi_NoGlobalThumb` reaches 453 ms (−38%), and `Downsampler_100x100` reaches 470 ms (−35%). `Jpeg_Q85` contributes a 25% reduction to 543 ms. The REST model shows weaker but consistent effects in the same direction: gaze-based ROI and downsampling reduce total pipeline latency by 9–12%, and the relative ordering across techniques is preserved. Techniques that discard task-relevant visual context or introduce high local overhead: `WebP_Lossy_Q85` and both YOLOv12 variants; increase latency rather than reduce it.

Local preprocessing overhead separates techniques into two well-separated clusters. Lightweight methods: JPEG compression, grayscale conversion, downsampling, thumbnail generation, and gaze-based cropping; add between 13 and 26 ms and account for at most 4.5% of total Realtime pipeline time. Their local cost is small enough that the server-side savings they enable survive intact in the end-to-end latency. Compute-heavy methods: WebP encoding, salient-region detection, and YOLOv12-based object detection; add between 42 and 94 ms, representing 7.6–12.4% of total pipeline time. This overhead is large enough to cancel or

reverse the server-side savings from their smaller payloads, which explains why these methods fail to reduce, and in some cases actively increase, end-to-end latency.

Measured preprocessing times for lightweight techniques appear 2–3× higher under the Realtime WebSocket interface than under REST, but this does not reflect additional computation. For tasks that complete in a few milliseconds, system-level event-loop scheduling jitter from the WebSocket connection dominates the measurement, adding apparent overhead without real local work. Compute-heavy techniques show near-parity across interfaces, because their runtime is dominated by the actual processing workload rather than by scheduling noise. A developer building a latency-sensitive pipeline can treat the lightweight cluster as practically free for pipeline budgeting purposes, and must treat the compute-heavy cluster as adding a fixed 42–94 ms overhead that has to be weighed against any server-side saving.

Payload size reduction depends on whether a technique changes compressed byte size alone or also changes decoded visual geometry. Compression-only methods: `Jpeg_Q85` and `WebP_Lossy_Q85`; achieve 50–60% reductions in serialized bytes while leaving the decoded pixel dimensions unchanged. Geometry-changing methods: downsampling, ROI cropping, and thumbnail generation; reduce payload by 85–100% by shrinking the spatial extent of the submitted image.

A smaller payload reliably predicts a shorter wait for the first server event for the Realtime model: the Spearman rank correlation between payload size and the request-sent-to-first-response interval is $\rho = 0.82$, while the weak Pearson correlation ($r = 0.14$) confirms the relationship is monotonic rather than proportional. This distinction matters because payload size correlates with transmission time but not with server-side processing time. A controlled calibration varying JPEG quality at fixed pixel dimensions — so that only serialized byte size changes — finds $R^2 \leq 0.02$ for the three REST providers and $R^2 = 0.10$ for `gpt-realtime-mini` (see section A.7). Server-side compute therefore dominates total pipeline latency for REST interfaces, and byte reduction via JPEG compression reduces transfer overhead without accelerating inference.

Token usage reduction follows a different mechanism from payload reduction: providers tokenize from the decoded pixel grid rather than from the compressed byte stream, so compressed file size does not determine token count. A developer applying `Jpeg_Q85` or `WebP_Lossy_Q85` can expect large byte savings while token usage remains unchanged across all four tested providers, because decoding restores the original pixel dimensions before tokenization. Only techniques that change decoded geometry — downsampling, ROI cropping, and thumbnail generation — can reduce token counts, and only when the resulting dimensions cross a provider-side tile or aspect-ratio threshold.

The magnitude of token reduction is provider-specific and discontinuous. `GazeRoi_NoGlobalThumb` reduced mean input tokens from 518 to 407 for `gpt-realtime-mini`; the same technique reduced tokens from 534 to 278 on `gpt-5.4-2026-03-05` and from 598 to 308 on `claude-sonnet-4-6`. `gemini-3-flash-preview` remained near its baseline of 1,372 tokens regardless of technique, because its flat per-tile accounting is insensitive to the tested resolutions. Adding a global thumbnail to an ROI method always increases token cost by inserting a second billed image part. Token savings cannot therefore be read as universally portable efficiency claims; the same technique can roughly halve input tokens on GPT-5.4 and Claude Sonnet while producing no saving on Gemini Flash 3. Token accounting should be verified per provider before projecting cost reductions.

Monetary cost translates the token results into a common currency, and the cost decomposition confirms that all preprocessing-driven savings appear exclusively in the visual input billing component. Output cost remains small and stable across all techniques, confirming that geometry reduction changes what is submitted but not what is generated. For compression-only methods, cost stays near the baseline despite large byte savings. For geometry-changing methods, cost reductions follow the same ordering as the token results, with no-thumbnail ROI variants and aggressive downsampling yielding the largest savings.

`gemini-3-flash-preview` is the exception: its near-flat token sensitivity means cost savings are negligible regardless of technique, making preprocessing selection irrelevant from a cost perspective on that provider. For the remaining three providers — `gpt-realtime-mini`, `gpt-5.4-2026-03-05`, and `claude-sonnet-4-6` — a provider-invariant practical rule emerges: any technique that reduces decoded visual geometry can reduce cost; any technique that only reduces compressed byte size cannot.

Table 7.1 consolidates baseline-relative efficiency and accuracy deltas for all preprocessing techniques in the Realtime Mini condition, serving as the primary cross-metric reference for the remainder of this chapter.

Table 7.1: Baseline-relative effect-size summary for the OpenAI Realtime Mini condition. Positive efficiency values indicate reductions relative to `Baseline_NoOp`; negative accuracy values indicate lower VQA-MHUG soft accuracy. Efficiency values are approximate, rounded from Figure 5.23; accuracy values are from the VQA-MHUG results in subsection 5.4.1. Sorted by accuracy Δ. **Bold green**: strong efficiency gain; mild green: moderate gain or near-zero accuracy change ($|\Delta| \leq 1$ pp); orange: moderate loss or efficiency decrease; **bold orange**: large accuracy loss (>15 pp).

Preprocessor	Latency Δ (%)	Payload Δ (%)	Tokens Δ (%)	Est. cost Δ (%)	Accuracy Δ (pp)
`Baseline_NoOp`	0	0	0	0	0
`Jpeg_Q85`	≈25	≈50	≈0	≈0	≈0
`WebP_Lossy_Q85`	−2	≈60	≈0	≈0	≈−1
`GazeRoi_NoGlobalThumb`	≈38	>85	>30	>20	−3
`GazeRoi_WithGlobalThumb`	≈21	≈70	≈15	≈10	−4
`Downsampler_100x100`	≈35	>95	>30	>20	−7
`Grayscale`	≈21	≈50	≈0	≈0	−7
`YoloV12SalientRoi_WithGlobalThumb`	−5	≈75	≈15	≈10	−7
`YoloV12SalientRoi`	−43	≈85	≈20	≈15	−8
`SalientRoi_WithGlobalThumb`	≈17	≈85	≈15	≈10	−22
`GlobalThumbOnly_28x28`	≈40	≈100	>30	>20	−25
`SalientRoi_NoGlobalThumb`	≈34	>95	>30	>20	−32

7.1.2. RQ2: Accuracy Impact of Preprocessing

Accuracy impact follows a single organizing principle: loss scales with how much task-relevant geometry is removed from the submitted image, not with how many bytes are saved. Compressing the byte representation using a quality of 85 without changing decoded pixel dimensions leaves accuracy intact; discarding spatial extent or colour channels reduces it. The accuracy Δ column of Table 7.1 records the baseline-relative change for each method under the Realtime condition.

Compression-only methods impose negligible accuracy loss. `Jpeg_Q85` reduces mean soft accuracy by 0.5 pp relative to `Baseline_NoOp`, and `WebP_Lossy_Q85` by 0.8 pp. Neither method removes decoded visual information, and both remain within measurement noise on the VQA-MHUG Realtime condition. The DriVQA and VOILA-A cross-checks confirm this: judge scores and quality ratings for both methods remain close to the baseline across all three datasets.

Geometry-changing methods that preserve task-relevant structure produce moderate but bounded losses. `GazeRoi_NoGlobalThumb` loses 3.1 pp, making it the strongest AR-specific accuracy trade-off: it achieves a 38% latency reduction and a payload reduction exceeding 85% while keeping the accuracy penalty small. `GazeRoi_WithGlobalThumb` loses 3.5 pp; adding the global thumbnail does not recover accuracy relative to the no-thumbnail variant despite increasing payload and token cost. `Downsampler_100x100` and `Grayscale` each lose approximately 7 pp, a visible but bounded penalty that may be acceptable when fine spatial detail or colour is not central to the task. Cross-provider comparison suggests that the `GazeRoi` penalty scales with model strength: on the GPT-5.4 REST provider, which achieves a baseline of ≈0.90, the drop reaches ≈10 pp — roughly three times the 3.1 pp observed on the Realtime model at baseline ≈0.75 — indicating that Gaze ROI should be re-evaluated on progressively stronger models as they become available.

Methods that discard task-relevant visual context cause large accuracy losses that disqualify them as general-purpose defaults. `GlobalThumbOnly_28x28` loses 24.5 pp and `SalientRoi_NoGlobalThumb` loses 31.1 pp, despite their strong efficiency profiles. YOLOv12-based variants lose 7–8 pp while also increasing end-to-end latency, offering no favourable trade-off on either dimension. A developer who must preserve accuracy should apply `Jpeg_Q85` when no geometry change is acceptable, and `GazeRoi_NoGlobalThumb` when AR-specific efficiency gains are required and a gaze signal is available.

7.1.3. RQ3: Overall Suitability by Deployment Scenario

Overall suitability depends on how an operator weighs the three competing objectives — latency, monetary cost, and accuracy — and a weighted scenario analysis across seven deployment profiles was applied to identify which techniques remain competitive across fundamentally different priority orderings (subsection 6.2.1). Because the scenario weights are illustrative rather than empirically grounded — no user study was conducted to elicit practitioner preferences — the analysis should be read as a structured sensitivity analysis: the key question is not which scenario scores highest, but whether any technique's rank remains stable as priorities shift.

Under the `gpt-realtime-mini` Realtime condition, `GazeRoi_NoGlobalThumb` emerges as the most robust single choice across the scenario space. It ranks first in six of the seven weighted scenarios and wins 64% of all 66 accuracy–latency–cost weight combinations enumerated from the three-objective simplex; `GlobalThumbOnly_28x28` wins 17%, `Jpeg_Q85` wins 12%, and the no-preprocessing baseline wins the remaining 8%. This robustness does not reflect a uniformly average profile across metrics: gaze-based ROI reaches the top of the Pareto frontier on both the latency–accuracy and cost–accuracy planes simultaneously (Figure 6.1, Figure 6.2). It combines a 38% latency reduction, a payload reduction exceeding 85%, and a cost reduction of approximately 20%, at an accuracy penalty of only 3 pp relative to `Baseline_NoOp`. Its first-place rank therefore holds under latency-first, cost-first, and balanced priorities, and also under accuracy-first weighting — because every competing geometry-reducing method incurs a larger accuracy penalty, making `GazeRoi_NoGlobalThumb` the strongest accuracy-preserving option among the techniques that also improve efficiency. The one scenario where it does not lead — quality-on-budget — is taken by `Jpeg_Q85`, which places the strongest relative weight on accuracy and accepts a lower efficiency gain.

`Jpeg_Q85` is the recommended fallback when gaze tracking is unavailable, unreliable, or when the deployment requires maximum accuracy preservation. It reduces latency by 25% and payload by roughly 50% while keeping VQA-MHUG accuracy within 0.5 pp of the unprocessed baseline — inside measurement noise in every tested model condition. Because it changes only the compressed byte representation and not the decoded pixel geometry, `Jpeg_Q85`'s accuracy preservation is robust across model tiers, providers, and datasets. It wins 12% of weight combinations, corresponding to quality-preserving priority settings, and is a strong second-place finisher across all seven scenarios.

Several other techniques are competitive on individual metrics but fail to translate those gains into general suitability. `GlobalThumbOnly_28x28` wins 17% of weight combinations under strongly efficiency-biased priorities, but its −24.5 pp accuracy loss and 52% per-sample non-determinism rate make it unsuitable as a deployment default. `Downsampler_100x100` is latency- and cost-efficient on the Realtime model, but exhibits a severe accuracy regression on stronger REST models (−23.5 pp on GPT-5.4), so it should not be treated as a general-purpose recommendation without per-model verification. The YOLOv12 variants and `WebP_Lossy_Q85` fail to win any weight combination: they increase total pipeline latency rather than reducing it, and therefore offer no favourable trade-off on the primary AR deployment criterion. Saliency-based methods without a global thumbnail achieve large payload and cost reductions, but their accuracy losses exceed 30 pp across all models and disqualify them as usable defaults under any scenario.

The cross-provider analysis qualifies the scope of the two-method recommendation. The conclusion — `GazeRoi_NoGlobalThumb` with gaze available, `Jpeg_Q85` without — holds for the Realtime Mini condition and is directionally consistent for GPT-5.4. For `claude-sonnet-4-6`, `Downsampler_100x100` leads in six of seven scenarios under the REST condition, because gaze ROI's server-side latency benefit is weaker there and the downsampler reaches comparable efficiency at a lower normalized accuracy cost. For `gemini-3-flash-preview`, the no-preprocessing baseline ranks first in six of seven scenarios: most preprocessing techniques add latency on that provider rather than reducing it, and its near-flat token accounting makes cost reduction from geometry changes negligible. These provider-level divergences reinforce the finding from RQ1 that provider and model selection dominates absolute pipeline performance and that preprocessing effectiveness is implementation-dependent. For interactive AR pipelines with reliable gaze tracking on Realtime-class models, `GazeRoi_NoGlobalThumb` is the recommended default; `Jpeg_Q85` is the safe fallback when gaze is unavailable or when accuracy constraints are binding.

7.2. Main Contributions

The contributions of this thesis are methodological, empirical, and practical.

Five-dimensional paired comparison and VQABench. The primary contribution is the first controlled, paired comparison of standard image preprocessing techniques measuring the joint effect on upload payload, provider-reported token usage, monetary cost, end-to-end pipeline latency, and VQA accuracy simultaneously within the same commercial cloud MLLM pipeline — the empirical instrument through which all three research questions are answered. As Table 3.1 shows, no prior benchmark treats preprocessing as an experimental variable while measuring its effect across all five dimensions; the upload payload column is absent from every prior commercial API study (Table 1.1), despite payload being the developer's first measurable consequence of choosing a preprocessing technique. The empirical dataset, experimental protocol, and results are released publicly as **VQABench**, a benchmark covering four provider-model configurations, three datasets, and 12 preprocessing techniques across more than 114,000 logged rows; its paired design and open release support independent replication and extension to additional techniques, providers, and model versions. A paper based on this work, *"How Much Does It Cost to Answer My Question? Benchmarking Preprocessing for Cloud VLM-based VQA"*, is submitted to the ACM Conference on Embedded Networked Sensor Systems (SenSys 2027) on 5 June 2026.

Paired black-box measurement methodology. The thesis introduces a reusable measurement methodology for evaluating commercial MLLM pipeline efficiency, providing the controlled evidence base required by all three research questions. The methodology uses a paired, same-sample design that treats the cloud provider as a black box, documents timing boundary definitions spanning preprocessing through response delivery, specifies token field mappings across four provider APIs, and dataset-appropriate evaluation metrics. Statistical reporting conventions — bootstrap 99% confidence intervals, effect-size reporting rather than significance testing — are documented to support reproducibility under external cloud variability.

Conceptual distinction between compression-only and geometry-changing preprocessing. Explaining the efficiency mechanisms behind RQ1 and establishing the basis for the accuracy preservation observed under RQ2, the thesis establishes a conceptual framework distinguishing two classes of preprocessing effect. Compression-only techniques reduce serialized bytes without changing the decoded pixel geometry, leaving provider-side token counts and cost unchanged. Geometry-changing techniques reduce both payload and token usage, but their token savings are provider-specific and discontinuous, because providers tokenize from the decoded pixel grid using tile-based rules.

Scenario-robust deployment recommendations for mobile AR pipelines. Addressing RQ3 directly, a weighted multi-criteria scenario analysis across seven deployment profiles produces a two-method recommendation that is stable across 64% of all accuracy–latency–cost weight combinations on the Realtime condition: `GazeRoi_NoGlobalThumb` for AR pipelines with reliable gaze tracking, and `Jpeg_Q85` as a safe fallback when gaze is unavailable or accuracy constraints are binding. Provider-level robustness checks confirm that this recommendation holds directionally for GPT-5.4, while divergent patterns on `claude-sonnet-4-6` and `gemini-3-flash-preview` demonstrate that preprocessing effectiveness depends on provider selection, reinforcing provider choice as the dominant system-level decision before preprocessing optimisation is considered.

7.3. Future Work

Adaptive and context-aware preprocessing selection. This thesis assumes that the visual input is always relevant to the posed question and that the same preprocessing strategy is applied to every query. Both assumptions can be relaxed. In practice, a user wearing AR glasses frequently asks questions that do not depend on the current visual scene — a navigation query, a factual lookup, or a reminder — and transmitting an irrelevant image in such cases wastes bandwidth and tokens without contributing to the answer. A lightweight

local relevance classifier or question-intent detector that determines, before transmission, whether the visual input is likely task-relevant could skip preprocessing and image transmission entirely for non-visual queries, producing efficiency gains that no preprocessing technique alone can achieve. Beyond binary relevance gating, different questions carry different visual information requirements: a fine-detail inspection query benefits from high-resolution context, while a broad object-identification query may be answered adequately from a compressed or cropped representation. Future work could therefore investigate supervised or rule-based routing mechanisms that map question type, image category, or gaze behaviour to a preprocessing strategy and evaluate whether per-query adaptive selection outperforms any fixed technique across a diverse workload.

Parameter sensitivity for preprocessing techniques. Most preprocessing techniques evaluated in this thesis use a single, fixed parameter configuration. However, many expose parameters that directly trade payload size against visual fidelity: the quality factor in JPEG compression, the ROI expansion factor and smoothing window in gaze-based cropping, or the resolution scale in downsampling. Systematic parameter sweeps across these technique families would characterise the Pareto frontier achievable within each family and determine whether parameter tuning can recover the accuracy losses observed in this study without sacrificing efficiency gains. This is particularly relevant for gaze-based ROI, where the 3.1 pp accuracy loss under `GazeRoi_NoGlobalThumb` suggests that the fixed ROI size and smoothing parameters do not optimally capture task-relevant regions for all samples.

Image classification-based stratification of preprocessing effects. The accuracy results in this thesis are aggregated across full datasets. Preprocessing effectiveness is likely to vary across image content types: high-detail technical images may be more sensitive to resolution reduction than open-scene photographs, and the gaze-ROI benefit may be larger for images where the attended region is clearly separable from the background. Stratifying preprocessing results by image category — using a standard classification model to label images before analysis — would reveal whether the efficiency-accuracy trade-offs reported here hold uniformly or are driven by specific content types. Such stratification would also provide empirical grounding for the adaptive routing strategy described above.

Combined preprocessing techniques. This thesis evaluates each preprocessing technique in isolation. Combining techniques that act on different pipeline stages — for example, geometry reduction followed by JPEG compression — may produce additive efficiency gains that neither technique achieves alone. A combined `Downsampler+Jpeg\_Q85` method, for instance, would reduce both the decoded pixel grid (lowering token count) and the serialized byte size (reducing transmission overhead), potentially matching the latency reduction of `GazeRoi\_NoGlobalThumb` without requiring a gaze signal. Future work should evaluate combination strategies and determine whether combined techniques extend the Pareto frontier established here or are outperformed by the single-technique methods already evaluated.

Specialized AR datasets with gaze, voice, and realistic capture conditions. None of the three datasets used in this thesis was collected specifically for wearable AR deployments. Real AR images exhibit quality degradation factors absent from studio-collected or web-sourced images: motion blur, narrow field-of-view cropping, lens distortion, and uncontrolled lighting variation. A dataset collected from wearable AR devices with simultaneous gaze tracking, voice-question recording, and real-world scene coverage would provide a more ecologically valid evaluation base for preprocessing research. Such a dataset would also enable investigation of the interaction between capture-quality degradation and preprocessing technique selection — a dimension that no existing public dataset supports.

Bibliography

[1] Michael Abrash. “Creating the Future: Augmented Reality, the next Human-Machine Interface”. In: *2021 IEEE International Electron Devices Meeting (IEDM)*. 2021, pp. 1–11. DOI: 10.1109/IEDM19574.2021.9720526.

[2] Lakshya Aggarwal et al. “From Videos to Conversations: Egocentric Instructions for Task Assistance”. In: *arXiv preprint arXiv:2602.01038* (2025). URL: https://arxiv.org/abs/2602.01038.

[3] Daghash K. Alqahtani, Muhammad Aamir Cheema, and Adel N. Toosi. “Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices”. In: *Service-Oriented Computing*. Ed. by Walid Gaaloul et al. Singapore: Springer Nature Singapore, 2025, pp. 142–150.

[4] Anthropic. *Claude Models Overview*. Accessed: 4 May 2026. 2026. URL: https://platform.claude.com/docs/en/about-claude/models/overview.

[5] Anthropic. *Pricing - Claude API Docs*. Accessed: 2026-04-01. 2026. URL: https://docs.anthropic.com/en/docs/about-claude/pricing.

[6] Anthropic. *Token counting - Claude API Docs*. Accessed: 2026-04-01. 2026. URL: https://docs.anthropic.com/en/docs/build-with-claude/token-counting.

[7] Anthropic. *Vision - Claude API Docs*. Accessed: 2026-04-01. 2026. URL: https://docs.anthropic.com/en/docs/build-with-claude/vision.

[8] Stanislaw Antol et al. “VQA: Visual Question Answering”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2425–2433.

[9] AssemblyAI. *Top APIs and Models for Real-Time Speech Recognition and Transcription in 2026*. Accessed: 2026-05-27. 2026. URL: https://www.assemblyai.com/blog/best-api-models-for-real-time-speech-recognition-and-transcription.

[10] Be My Eyes. *Be My Eyes: Lend Your Eyes to the Blind*. Mobile application. 2015. URL: https://www.bemyeyes.com/.

[11] Pete Beckman et al. “Benchmarking the Effects of Operating System Interference on Extreme-Scale Parallel Machines”. In: *Cluster Computing* 11.1 (2008), pp. 3–16. DOI: 10.1007/s10586-007-0047-2.

[12] Brian N. Bershad, Richard P. Draves, and Alessandro Forin. “Using Microbenchmarks to Evaluate System Performance”. In: *Proceedings of the IEEE*. 1992.

[13] Riccardo Bovo et al. “EmBARDiment: An Embodied AI Agent for Productivity in XR”. In: *arXiv preprint arXiv:2408.08158* (2024). URL: https://arxiv.org/abs/2408.08158.

[14] Dongwook Cha et al. “Designing Memory-Augmented AR Agents for Spatiotemporal Reasoning in Personalized Task Assistance”. In: *arXiv preprint arXiv:2508.08774* (2025). URL: https://arxiv.org/abs/2508.08774.

[15] Eun Chang, Rohit Patel, et al. “WearVQA: A Visual Question Answering Benchmark for Wearables in Egocentric Authentic Real-world Scenarios”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. arXiv:2511.22154. 2025. URL: https://arxiv.org/abs/2511.22154.

[16] Liang Chen et al. “An Image is Worth 1/2 Tokens After Layer 2: Plug-and-Play Inference Acceleration for Large Vision-Language Models”. In: *European Conference on Computer Vision*. 2024. URL: https://arxiv.org/abs/2403.06764.

[17] Qinyu Chen and Jiawen Qi. *Eye Gaze Tells You Where to Compute: Gaze-Driven Efficient VLMs*. Sept. 2025. DOI: 10.48550/arXiv.2509.16476. arXiv: 2509.16476 [cs.CV].

[18] Qinyu Chen and Jiawen Qi. *Eye Gaze Tells You Where to Compute: Gaze-Driven Efficient VLMs*. 2025. arXiv: 2509.16476 [cs.CV]. URL: https://arxiv.org/abs/2509.16476.

[19] Wanyin Cheng and Zanxi Ruan. “BLaVe-CoT: Consistency-Aware Visual Question Answering for Blind and Low Vision Users”. In: *arXiv preprint arXiv:2509.06010* (2025). URL: https://arxiv.org/abs/2509.06010.

[20] Joong Ho Choi et al. “CompactPrompt: A Unified Pipeline for Prompt and Data Compression in LLM Workflows”. In: *Proceedings of The 2nd Workshop on LLMs and Generative AI for Finance (ACM ICAIF '25)*. 2025. URL: https://arxiv.org/abs/2510.18043.

[21] Joong Ho Choi et al. *Token-Efficient Multimodal Reasoning via Image Prompt Packaging*. 2026. arXiv: 2604.02492 [cs.CV]. URL: https://arxiv.org/abs/2604.02492.

[22] Hannah Christodoulou, Hassan Sajjad, Fahim Dalvi, et al. “DARE: Diverse Visual Question Answering with Robustness Evaluation”. In: *Transactions of the Association for Computational Linguistics (TACL)* (2025). Author list incomplete; check official record. DOI: 10.1162/TACL.a.29. URL: https://direct.mit.edu/tacl/article/doi/10.1162/TACL.a.29/133041/.

[23] Xiangxiang Chu et al. “MobileVLM: A Fast, Strong and Open Vision Language Assistant for Mobile Devices”. In: *arXiv preprint arXiv:2312.16886* (2023). URL: https://arxiv.org/abs/2312.16886.

[24] Zhihao Chu et al. “nnPerf: Demystifying DNN Runtime Inference Latency on Mobile Platforms”. In: *Proceedings of the ACM Conference on Embedded Networked Sensor Systems*. 2023. DOI: 10.1145/3625687.3625797.

[25] Cody Coleman et al. “Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark”. In: *ACM SIGOPS Operating Systems Review*. Vol. 53. 1. 2019, pp. 14–25. DOI: 10.1145/3352020.3352024.

[26] Microsoft Corporation. *HoloLens 2 hardware specification*. https://learn.microsoft.com/en-us/hololens/hololens2-hardware. 2023.

[27] Abhishek Das et al. “Human Attention in Visual Question Answering: Do Humans and Deep Networks Look at the Same Regions?” In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 932–937. URL: https://arxiv.org/abs/1606.05589.

[28] Chenyou Fan. “EgoVQA—An Egocentric Video Question Answering Benchmark Dataset”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 2019, pp. 4359–4366. DOI: 10.1109/ICCVW.2019.00536.

[29] Tharindu Fernando et al. “Task Specific Visual Saliency Prediction with Memory Augmented Conditional Generative Adversarial Networks”. In: *arXiv preprint arXiv:1803.03354* (2018). arXiv: 1803.03354 [cs.CV]. URL: https://arxiv.org/abs/1803.03354.

[30] Geeky Gadgets. *Apple Smart Glasses Leak: Visual Intelligence & Gemini Siri*. Geeky Gadgets. Accessed 2026-05-11. 2026. URL: https://www.geeky-gadgets.com/apple-glasses-final-testing/.

[31] Geeky Gadgets. *Google vs Meta Smart Glasses: Which AI Frames Are Better*. Geeky Gadgets. Accessed 2026-05-11. 2026. URL: https://www.geeky-gadgets.com/google-vs-meta-smart-glasses/.

[32] Giuseppe Ginesu, Ruggero Pintus, and Daniele D. Giusto. “Objective assessment of the WebP image coding algorithm”. In: *Signal Processing: Image Communication* 27.8 (2012), pp. 867–877. DOI: 10.1016/j.image.2012.06.002.

[33] Google. *Billing | Gemini API*. Accessed: 2026-04-01. 2026. URL: https://ai.google.dev/gemini-api/docs/billing.

[34] Google. *Gemini API Models*. Accessed: 4 May 2026. 2026. URL: https://ai.google.dev/gemini-api/docs/models.

[35] Google. *Gemini Developer API pricing*. Accessed: 2026-04-01. 2026. URL: https://ai.google.dev/gemini-api/docs/pricing.

[36] Google. *Image understanding | Gemini API*. Accessed: 2026-04-01. 2026. URL: https://ai.google.dev/gemini-api/docs/image-understanding.

[37] Google. *Understand and count tokens | Gemini API*. Accessed: 2026-04-01. 2026. URL: https://ai.google.dev/gemini-api/docs/tokens.

[38] Google DeepMind. *Project Astra: A Research Prototype Exploring the Future of AI Assistants*. Google DeepMind. 2024. URL: https://deepmind.google/models/project-astra/.

[39] Yash Goyal et al. “Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017.

[40] Danna Gurari et al. “VizWiz Grand Challenge: Answering Visual Questions from Blind People”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 3608–3617.

[41] Xiaodi Hou and Liqing Zhang. “Saliency Detection: A Spectral Residual Approach”. In: vol. 2007. June 2007. DOI: 10.1109/CVPR.2007.383267.

[42] Mina Huh, Yi-Hao Peng, and Amy Pavel. “Long-Form Answers to Visual Questions from Blind and Low Vision People”. In: *Conference on Language Modeling (COLM)*. 2024. URL: https://www.yihaopeng.tw/pdf/COLM24_VizwizLF.pdf.

[43] Kotaro Inoue. *Context-Independent OCR with Multimodal LLMs: Effects of Image Resolution and Visual Complexity*. 2025. arXiv: 2503.23667 [cs.CV]. URL: https://arxiv.org/abs/2503.23667.

[44] Introl. *Voice AI Infrastructure: Building Real-Time Speech Agents*. Accessed: 2026-05-27. 2026. URL: https://introl.com/blog/voice-ai-infrastructure-real-time-speech-agents-asr-tts-guide-2025.

[45] Hyeong-GI Jeon and Kyoung-Hee Lee. “Region-of-Interest Extraction Method to Increase Object-Detection Performance in Remote Monitoring System”. In: *Applied Sciences* 15.10 (2025), p. 5328. DOI: 10.3390/app15105328.

[46] Huiqiang Jiang et al. “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models”. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 13358–13376. URL: https://arxiv.org/abs/2310.05736.

[47] Zhuohang Jiang et al. “SUPERGLASSES: Benchmarking Vision Language Models as Intelligent Agents for AI Smart Glasses”. In: *arXiv preprint arXiv:2602.22683* (2025). URL: https://arxiv.org/abs/2602.22683.

[48] Christopher Kanan and Garrison W. Cottrell. “Color-to-Grayscale: Does the Method Matter in Image Recognition?” In: *PLOS ONE* 7.1 (2012), e29740. DOI: 10.1371/journal.pone.0029740.

[49] Rajan Kannan et al. “Beyond Inference: Performance Analysis of DNN Server Overheads for Computer Vision”. In: *arXiv preprint arXiv:2403.12981* (2024).

[50] Chia-Hao Kao et al. *Bridging Compressed Image Latents and Multimodal Large Language Models*. 2025. arXiv: 2407.19651 [cs.CV]. URL: https://arxiv.org/abs/2407.19651.

[51] Cornelius Kummer et al. *Prompt Compression in the Wild: Measuring Latency, Rate Adherence, and Quality for Faster LLM Inference*. 2026. arXiv: 2604.02985 [cs.IR]. URL: https://arxiv.org/abs/2604.02985.

[52] Stefanos Laskaridis et al. “MELTing Point: Mobile Evaluation of Language Transformers”. In: *Proceedings of the ACM International Conference on Mobile Computing and Networking*. 2024. DOI: 10.1145/3636534.3690668.

[53] Dong-U Lee et al. “Energy-Efficient Image Compression for Resource-Constrained Platforms”. In: *IEEE Transactions on Image Processing* 18.9 (2009), pp. 2100–2113. DOI: 10.1109/TIP.2009.2022438.

[54] Jaewook Lee et al. “GazePointAR: A Context-Aware Multimodal Voice Assistant for Pronoun Disambiguation in Wearable Augmented Reality”. In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems (CHI)*. 2024, pp. 1–20. DOI: 10.1145/3613904.3642230.

[55] Jaewook Lee et al. “Guided Reality: Generating Visually-Enriched AR Task Guidance with LLMs and Vision Models”. In: *arXiv preprint arXiv:2508.03547* (2025). URL: https://arxiv.org/abs/2508.03547.

[56] Chenxu Li et al. *Res-Bench: Benchmarking the Robustness of Multimodal Large Language Models to Dynamic Resolution Input*. 2025. arXiv: 2510.16926 [cs.CV]. URL: https://arxiv.org/abs/2510.16926.

[57] Yanhong Li, Ying Shan, and Jungong Han. *Text or Pixels? It Takes Half: On the Token Efficiency of Visual Text Inputs in Multimodal LLMs*. 2025. arXiv: 2510.18279 [cs.CV]. URL: https://arxiv.org/abs/2510.18279.

[58] Yijun Liang et al. *ColorBench: Can VLMs See and Understand the Colorful World? A Comprehensive Benchmark for Color Perception, Reasoning, and Robustness*. 2025. arXiv: 2504.10514 [cs.CV]. URL: https://arxiv.org/abs/2504.10514.

[59] Shihan Liu et al. "EdgeYOLO: An Edge-Real-Time Object Detector". In: *2023 42nd Chinese Control Conference (CCC)*. IEEE. 2023, pp. 7507–7512. DOI: 10.23919/CCC58697.2023.10239786.

[60] Yuan Liu et al. "MMBench: Is Your Multi-modal Model an All-around Player?" In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2024. URL: https://arxiv.org/abs/2307.06281.

[61] Zechun Liu et al. "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases". In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 2024. arXiv: 2402.14905 [cs.LG]. URL: https://arxiv.org/abs/2402.14905.

[62] Michael A. Lones. "Avoiding common machine learning pitfalls". In: *Patterns* 5.10 (Oct. 2024). ISSN: 2666-3899. DOI: 10.1016/j.patter.2024.101046. URL: https://doi.org/10.1016/j.patter.2024.101046.

[63] Dongchen Lu et al. *InternVL-X: Advancing and Accelerating InternVL Series with Efficient Visual Token Compression*. 2025. arXiv: 2503.21307 [cs.CV]. URL: https://arxiv.org/abs/2503.21307.

[64] Pan Lu et al. "MathVista: Evaluating Mathematical Reasoning of Foundation Models in Visual Contexts". In: *International Conference on Learning Representations (ICLR)*. 2024. URL: https://arxiv.org/abs/2310.02255.

[65] Zhenyan Lu et al. "Demystifying small language models for edge deployment". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2025, pp. 14747–14764.

[66] Kenneth Marino et al. "OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 3195–3204. DOI: 10.1109/CVPR.2019.00331.

[67] Pierre Marza et al. *An experimental study of the vision-bottleneck in VQA*. 2022. arXiv: 2202.06858 [cs.CV]. URL: https://arxiv.org/abs/2202.06858.

[68] Junayed Merchant, Li Liu, et al. "Exploring the Use of VLMs for Navigation Assistance for People with Blindness and Low Vision". In: *arXiv preprint arXiv:2603.15624* (2024). URL: https://arxiv.org/abs/2603.15624.

[69] Meta and Ray-Ban. *Ray-Ban Meta AI Glasses Gen 2 & Gen 1*. Product page. Accessed 2026-05-11. 2025. URL: https://www.ray-ban.com/usa/ray-ban-meta-ai-glasses.

[70] Antje S. Meyer, Astrid M. Sleiderink, and Willem J. M. Levelt. "Viewing and naming objects: Eye movements during noun phrase production". In: *Cognition* 66.2 (1998), B25–B33. DOI: 10.1016/S0010-0277(98)00009-3.

[71] Akos Nagy, Yannis Spyridis, and Vasileios Argyriou. "Cross-Format Retrieval-Augmented Generation in XR with LLMs for Context-Aware Maintenance Assistance". In: *arXiv preprint arXiv:2502.15604* (2025). URL: https://arxiv.org/abs/2502.15604.

[72] Jing Bi Nguyen et al. "MISAR: A Multimodal Instructional System with Augmented Reality". In: *arXiv preprint arXiv:2310.11699* (2023). URL: https://arxiv.org/abs/2310.11699.

[73] Minh T. Nguyen and Thuong TK. Nguyen. "Advanced and AI Embedded Technologies in Education: Effectiveness, Recent Developments, and Opening Issues". In: *Journal of Future Artificial Intelligence and Technologies* 1.3 (Oct. 2024), pp. 191–200. DOI: 10.62411/faith.3048-3719-19. URL: https://faith.futuretechsci.org/index.php/FAITH/article/view/19.

[74] N. E. Nwogbaga et al. "Investigation into the effect of data reduction in offloadable task for distributed IoT-fog-cloud computing". In: *Journal of Cloud Computing* 10.1 (2021). DOI: 10.1186/s13677-021-00254-6.

[75] OpenAI. *API Reference Overview*. Accessed: 2026-05-20. 2026. URL: https://developers.openai.com/api/reference/overview.

[76] OpenAI. *gpt-realtime-mini Model*. Accessed: 2026-05-07. 2026. URL: https://developers.openai.com/api/docs/models/gpt-realtime-mini.

[77] OpenAI. *Images and vision*. Accessed: 2026-04-01. 2026. URL: https://developers.openai.com/api/docs/guides/images-vision.

[78] OpenAI. *Managing costs*. Accessed: 2026-04-01. 2026. URL: https://developers.openai.com/api/docs/guides/realtime-costs/.

[79] OpenAI. *Pricing*. Accessed: 2026-05-07. 2026. URL: https://openai.com/api/pricing.

[80] OpenAI. *Realtime API Guide*. Accessed: 2026-05-20. 2026. URL: https://developers.openai.com/api/docs/guides/realtime.

[81] OpenAI. *Realtime API WebSocket Guide*. Accessed: 2026-05-20. 2026. URL: https://developers.openai.com/api/docs/guides/realtime-websocket.

[82] OpenAI. *Realtime API with WebRTC*. Accessed: 2026. 2026. URL: https://platform.openai.com/docs/guides/realtime-webrtc.

[83] OpenAI. *Realtime API with WebSocket*. Accessed: 2026. 2026. URL: https://platform.openai.com/docs/guides/realtime-websocket.

[84] OpenAI. *What are tokens and how to count them?* Accessed: 2026-05-20. 2026. URL: https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them.

[85] Tianfan Peng et al. *Can Visual Input Be Compressed? A Visual Token Compression Benchmark for Large Multimodal Models*. 2025. arXiv: 2511.02650 [cs.CV]. URL: https://arxiv.org/abs/2511.02650.

[86] Jiaxu Qian et al. *Zoomer: Adaptive Image Focus Optimization for Black-box MLLM*. 2025. arXiv: 2505.00742 [cs.CV]. URL: https://arxiv.org/abs/2505.00742.

[87] Vijay Janapa Reddi et al. "MLPerf Inference Benchmark". In: *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*. 2020. DOI: 10.1109/ISCA45697.2020.00045.

[88] David Georg Reichelt, Reiner Jung, and André van Hoorn. "Overhead Measurement Noise in Different Runtime Environments". In: *arXiv preprint arXiv:2411.05491* (2024).

[89] Kaavya Rekanar et al. "DriVQA: A gaze-based dataset for visual question answering in driving scenarios". In: *Data in Brief* 59 (2025), p. 111367. DOI: 10.1016/j.dib.2025.111367. URL: https://data.mendeley.com/datasets/p25744hwrc/1.

[90] Jinke Ren et al. "An Edge-Computing Based Architecture for Mobile Augmented Reality". In: *IEEE Network* 33.4 (2019), pp. 162–169. DOI: 10.1109/MNET.2018.1800132.

[91] Rokid. *Rokid AI & AR Glasses—Redefining Reality*. Product page. Accessed 2026-05-11. 2025. URL: https://global.rokid.com/.

[92] D. Sahu et al. "Edge assisted energy optimization for mobile AR applications". In: *Scientific Reports* 15.1 (2025), p. 93731. DOI: 10.1038/s41598-025-93731-w.

[93] Yuzhang Shang et al. *LLaVA-PruMerge: Adaptive Token Reduction for Efficient Large Multimodal Models*. 2024. arXiv: 2403.15388 [cs.CV]. URL: https://arxiv.org/abs/2403.15388.

[94] Susmit Shannigrahi, Spyridon Mastorakis, and Francisco R. Ortega. "Next-Generation Networking and Edge Computing for Mixed Reality Real-Time Interactive Systems". In: *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2020, pp. 1–6. DOI: 10.1109/ICCWorkshops49005.2020.9145075.

[95] Kele Shao et al. *A Survey of Token Compression for Efficient Multimodal Large Language Models*. 2025. arXiv: 2507.20198 [cs.CV]. URL: https://arxiv.org/abs/2507.20198.

[96] Ahmed Sharshar et al. *Vision-Language Models for Edge Networks: A Comprehensive Survey*. 2025. arXiv: 2502.07855 [cs.CV]. URL: https://arxiv.org/abs/2502.07855.

[97] Ryan Shea et al. “Towards Fully Offloaded Cloud-based AR: Design, Implementation and Experience”. In: *Proceedings of the 8th ACM on Multimedia Systems Conference*. MMSys'17. Taipei, Taiwan: Association for Computing Machinery, 2017, pp. 321–330. ISBN: 9781450350020. DOI: 10.1145/3083187.3084012. URL: https://doi.org/10.1145/3083187.3084012.

[98] Yuling Shi, Xiaodong Gu, et al. *CodeOCR: On the Effectiveness of Vision Language Models in Code Understanding*. 2026. arXiv: 2602.01785 [cs.CV]. URL: https://arxiv.org/abs/2602.01785.

[99] Wooseok Shin et al. *Prompt Compression with Context-Aware Sentence Encoding for Fast and Improved LLM Inference*. 2024. arXiv: 2409.01227 [cs.CL]. URL: https://arxiv.org/abs/2409.01227.

[100] Gaurav Shinde et al. *A Survey on Efficient Vision-Language Models*. 2025. arXiv: 2504.09724 [cs.CV]. URL: https://arxiv.org/abs/2504.09724.

[101] David Skinner and William Kramer. “Understanding the Causes of Performance Variability in HPC Workloads”. In: *Proceedings of the IEEE International Workload Characterization Symposium*. IEEE. 2005, pp. 137–149.

[102] Smallest.ai. *Google Cloud Speech-to-Text in 2026: Still Worth It?* Accessed: 2026-05-27. 2026. URL: https://smallest.ai/blog/is-google-cloud-speech-to-text-still-the-right-choice.

[103] Ekta Sood et al. “VQA-MHUG: A Gaze Dataset to Study Multimodal Neural Attention in Visual Question Answering”. In: *CoRR* abs/2109.13116 (2021). arXiv: 2109.13116. URL: https://arxiv.org/abs/2109.13116.

[104] Hossein Talebi and Peyman Milanfar. “Learning To Resize Images for Computer Vision Tasks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 497–506.

[105] TCL RayNeo. *RayNeo X2 Waveguide Glasses*. Product page. Accessed 2026-05-11. 2025. URL: https://www.rayneo.com/products/tcl-rayneo-x2.

[106] The Next Web. *Apple Testing Four Frame Designs for AI Smart Glasses Ahead of 2027 Launch*. The Next Web. Accessed 2026-05-11. 2026. URL: https://thenextweb.com/news/apple-smart-glasses-four-frame-designs-testing-2027.

[107] Philippe Thévenaz, Thierry Blu, and Michael Unser. “Image Interpolation and Resampling”. In: *Handbook of Medical Imaging, Processing and Analysis*. 2000.

[108] Gregory K. Wallace. “The JPEG still picture compression standard”. In: *Communications of the ACM* 34.4 (1991), pp. 30–44. DOI: 10.1145/103085.103089.

[109] Jiaqi Wang et al. “CRAG-MM: Multi-modal Multi-turn Comprehensive RAG Benchmark”. In: *arXiv preprint arXiv:2510.26160* (2025). URL: https://arxiv.org/abs/2510.26160.

[110] Xin Wang et al. “HoloAssist: An Egocentric Human Interaction Dataset for Interactive AI Assistants in the Real World”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2023. URL: https://arxiv.org/abs/2309.17024.

[111] B. Wong, Joya Chen, Mike Zheng Shou, et al. “Generating Dialogues from Egocentric Instructional Videos for Task Assistance: Dataset, Method and Benchmark”. In: *arXiv preprint arXiv:2508.11192* (2025). URL: https://arxiv.org/abs/2508.11192.

[112] Bo Xu et al. “Semantic Distillation Guided Salient Object Detection”. In: *arXiv preprint arXiv:2203.04076* (2022). OPPO Research Institute. arXiv: 2203.04076 [cs.CV]. URL: https://arxiv.org/abs/2203.04076.

[113] Fang Xu et al. “Integrating augmented reality and LLM for enhanced cognitive support in critical audio communications”. In: *International Journal of Human-Computer Studies* 194 (2025), p. 103402. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2024.103402. URL: https://www.sciencedirect.com/science/article/pii/S107158192400185X.

[114] Kun Yan et al. “Voila-A: Aligning Vision-Language Models with User's Gaze Attention”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson et al. Vol. 37. Curran Associates, Inc., 2024, pp. 1890–1918. DOI: 10.52202/079017-0060. URL: https://proceedings.neurips.cc/paper_files/paper/2024/file/03738e5f26967582eeb3b57eef82f1f0-Paper-Conference.pdf.

[115] Shukang Yin et al. *A Survey on Multimodal Large Language Models*. 2023. arXiv: 2306.13549 [cs.CV]. URL: https://arxiv.org/abs/2306.13549.

[116] Shukang Yin et al. “A survey on multimodal large language models”. In: *National Science Review* 11.12 (2024). Review article, nwae403. ISSN: 2095-5138. DOI: 10.1093/nsr/nwae403. URL: https://doi.org/10.1093/nsr/nwae403.

[117] Ayman Younis, Brian Qiu, and Dario Pompili. “Latency-aware Hybrid Edge Cloud Framework for Mobile Augmented Reality Applications”. In: June 2020, pp. 1–9. DOI: 10.1109/SECON48991.2020.9158429.

[118] Xiang Yue et al. “MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024, pp. 9556–9567. DOI: 10.1109/CVPR52733.2024.00913.

[119] Jiarui Zhang et al. “Visual Cropping Improves Zero-Shot Question Answering of Multimodal Large Language Models”. In: *R0-FoMo:Robustness of Few-shot and Zero-shot Learning in Large Foundation Models*. 2023. URL: https://openreview.net/forum?id=YrYcoV2dAk.

[120] Yi Zhao et al. *“Less is More”: Reducing Cognitive Load and Task Drift in Real-Time Multimodal Assistive Agents for the Visually Impaired*. 2025. arXiv: 2511.00945 [cs.HC]. URL: https://arxiv.org/abs/2511.00945.

[121] Zihui Zhao, Yingxin Li, and Yang Li. *LFTR: Learning-Free Token Reduction for Multimodal Large Language Models*. 2025. arXiv: 2501.17391 [cs.CV]. URL: https://arxiv.org/abs/2501.17391.

[122] Baichuan Zhou et al. “TinyLLaVA: A Framework of Small-scale Large Multimodal Models”. In: *arXiv preprint arXiv:2402.14289* (2024). arXiv: 2402.14289 [cs.LG]. URL: https://arxiv.org/abs/2402.14289.

A

Appendix A

A.1. Token Cost Calculation

The total cost per request is computed from the token fields described in Table 4.4 using the per-provider rates listed in Table 4.5 [79, 78, 76, 35, 33, 37, 5, 6]. Three token quantities are derived first:

$$\texttt{cached\_input\_tokens} = \texttt{cached\_tokens}, \tag{A.1}$$

$$\texttt{uncached\_text\_tokens} = \max(\texttt{text\_input\_tokens} - \texttt{cached\_input\_tokens}, 0), \tag{A.2}$$

$$\texttt{uncached\_image\_tokens} = \texttt{image\_input\_tokens}. \tag{A.3}$$

These are then combined with the per-token rates $p_{\text{text-in}}$, $p_{\text{image-in}}$, $p_{\text{cached-in}}$, and p_{out} (all in USD per 10^6 tokens) to obtain the total cost:

$$\begin{aligned}\texttt{total\_cost\_usd} = {} & \frac{\texttt{uncached\_text\_tokens}}{10^6} p_{\text{text-in}} + \frac{\texttt{uncached\_image\_tokens}}{10^6} p_{\text{image-in}} \\ & + \frac{\texttt{cached\_input\_tokens}}{10^6} p_{\text{cached-in}} + \frac{\texttt{output\_tokens}}{10^6} p_{\text{out}}.\end{aligned} \tag{A.4}$$

For providers that do not expose a modality split (Gemini and Anthropic), `text_input_tokens` falls back to a copy of `input_tokens` and `image_input_tokens` is set to zero. In that case the formula reduces to charging `input_tokens` − `cached_tokens` at the ordinary input rate, `cached_tokens` at the cached-input rate, and `output_tokens` at the output rate. This convention makes cached-token accounting explicit and prevents cached input from being counted once at the ordinary input rate and again at the cached rate.

A.2. Task-Performance Metric Fields

Table A.1 lists all task-performance metric fields produced by the evaluation procedures described in section 2.5, grouped by dataset and evaluation technique. All fields are recorded per sample. VQA-MHUG fields reflect the outcome of a single model prediction against the human reference answers. DriVQA and VOILA-A

fields are produced by LLM-based judge evaluations applied to each individual response; absolute fields capture the judge's quality assessment of that response, and pairwise fields record the per-sample comparison outcome against `Baseline_NoOp`.

Table A.1: Task-performance metric fields recorded per dataset. Fields are grouped by dataset and evaluation technique.

Field	Description
VQA-MHUG — VQA-style soft accuracy	
`reference_answer`	Ground-truth answer string, unnormalized.
`normalized_prediction`	Model answer after applying the normalization pipeline.
`exact_match`	Binary indicator of whether the normalized prediction matches one of the human reference answers.
`vqa_accuracy`	Per-sample VQA soft-accuracy score.
DriVQA — LLM judge: human-distribution rubric (absolute and pairwise)	
`reference_answer`	Ground-truth answer string for the sample.
`assistant_text`	Raw response text returned by the model.
`drivqa_evaluation_protocol`	Evaluation protocol identifier used for this sample.
`drivqa_judge_model`	Judge model used for absolute and pairwise evaluation.
`drivqa_human_answer_groups`	Human answer distribution used as the judge reference.
`drivqa_weighted_match_score`	Per-sample weighted match score against the human-answer distribution.
`drivqa_majority_alignment`	Per-sample binary majority-alignment indicator.
`drivqa_minor_valid_alignment`	Per-sample binary minor-valid-alignment indicator.
`drivqa_contradiction_rate`	Per-sample contradiction rate assigned by the judge.
`drivqa_ambiguity_handling`	Per-sample ambiguity-handling score assigned by the judge.
`drivqa_overall_score`	Per-sample composite judge quality score; primary DriVQA quality measure.
`drivqa_major_error_type`	Diagnostic label for the dominant error type, where applicable.
`drivqa_short_reason`	Short judge-assigned reason string.
`drivqa_absolute_error`	Error flag for a failed absolute judge evaluation.
`drivqa_pairwise_winner_vs_baseline`	Per-sample pairwise comparison result against `Baseline_NoOp`.
`drivqa_pairwise_confidence`	Judge confidence for the pairwise comparison outcome.
`drivqa_pairwise_reason`	Short reason string for the pairwise comparison outcome.
`drivqa_pairwise_error`	Error flag for a failed pairwise evaluation.
VOILA-A — LLM judge: open-ended response quality (absolute and pairwise)	
`reference_answer`	Ground-truth answer string used as the minimum-correctness anchor.
`assistant_text`	Raw response text returned by the model.
`voila_evaluation_protocol`	Evaluation protocol identifier used for this sample.
`voila_scene_context`	Scene description or image caption passed to the judge.
`voila_judge_model`	Judge model used for absolute and pairwise evaluation.
`voila_evidence_consistency`	Per-sample evidence-consistency criterion score (0–5).
`voila_coverage`	Per-sample coverage criterion score (0–5).
`voila_detail_quality`	Per-sample detail-quality criterion score (0–5).
`voila_unsupported_claim_control`	Per-sample unsupported-claim control criterion score (0–5).
`voila_overall_score`	Per-sample composite judge quality score; primary VOILA-A quality measure.
`voila_evidence_status`	Per-sample evidence-status label assigned by the judge.
`voila_major_error_type`	Diagnostic label for the dominant error type, where applicable.
`voila_short_reason`	Short judge-assigned reason string.
`voila_absolute_error`	Error flag for a failed absolute judge evaluation.
`voila_pairwise_winner_vs_baseline`	Per-sample pairwise comparison result against `Baseline_NoOp`.
`voila_pairwise_confidence`	Judge confidence for the pairwise comparison outcome.
`voila_pairwise_reason`	Short reason string for the pairwise comparison outcome.
`voila_pairwise_error`	Error flag for a failed pairwise evaluation.

A.3. Prompt Instruction Strings

The three system instruction strings used across datasets are listed below. `VQA\_SHORT\_ANSWER\_INSTRUCTIONS` is used for VQA-MHUG; it enforces a strict one- to two-word lowercase output format. `DRIVQA\_OPEN\_RESPONSE\_INSTRUCTIONS` is used for DriVQA; it requests a single concise sentence in a driving-scene context. `VOILA\_OPEN\_RESPONSE\_INSTRUCTIONS` is used for VOILA-A; it allows a full sentence without the short VQA format constraint.

```
VQA_SHORT_ANSWER_INSTRUCTIONS = (
    "You are answering a VQA benchmark. Output only the canonical short answer that humans
    ↪  would write.\n\n"
    "Rules:\n"
    "- Output 1 to 2 words (max 3).\n"
    "- No full sentence, no extra objects, no prepositions like 'on', 'in', 'with'.\n"
    "- For action questions ('What is X doing?'), output only the main verb (e.g., 'writing',
    ↪  'eating', 'standing').\n"
```

```
    "- No articles (a/an/the), no punctuation.\n"
    "- Lowercase.\n"
    "- Numbers as digits.\n"
    "- If unsure, choose the simplest, most common answer.\n\n"
    "Examples:\n"
    "Q: what is the man doing? A: writing\n"
    "Q: what material is the counter made of? A: wood\n"
    "Q: does the laptop have a cable? A: yes\n"
    "Q: how many people have a camera? A: 1"
)

DRIVQA_OPEN_RESPONSE_INSTRUCTIONS = (
    "You are a visual assistant for driving-scene visual question answering. "
    "Answer the user's question using the image content. "
    "Provide a direct, helpful natural-language answer of 1 sentence. "
    "Keep the response concise, but do not force it into a one- or two-word VQA format."
)

VOILA_OPEN_RESPONSE_INSTRUCTIONS = (
    "You are a visual assistant. Answer the user's question using the image content."
    "Provide a direct, helpful natural-language answer of 1 sentence."
    "Do not force the answer into a one- or two-word VQA format. Keep the response concise
    ↪ unless the question clearly needs more detail."
)
```

A.4. Gaze Preprocessing: Timing Assumptions and ROI Construction

The recording window for each gaze trace is inherited from the source dataset and is not redefined by the thesis pipeline. The experiment therefore does not model how long a live AR system would need to observe gaze before sending an image to the cloud. A realistic deployment could instead start collecting gaze as soon as the user begins speaking, accumulate points during the utterance, and apply preprocessing before transmitting. The present experiment approximates that interaction using already-recorded gaze traces, but does not evaluate the latency or behavioural effects of the gaze-recording window itself.

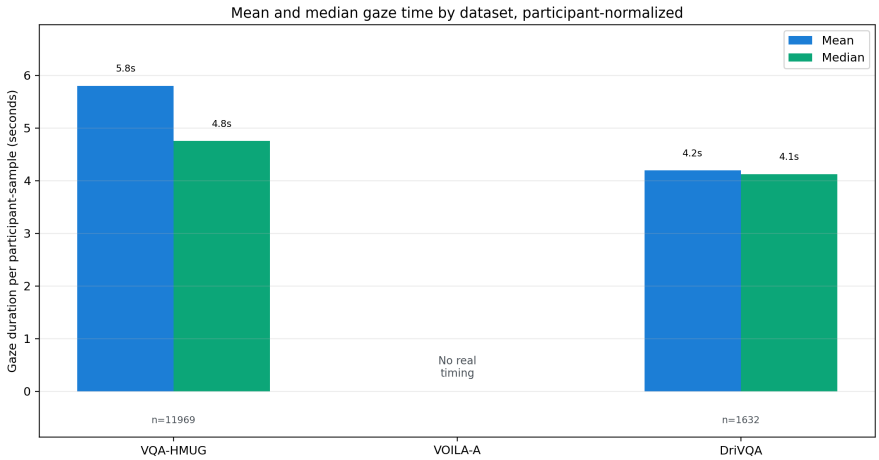


Figure A.1: Mean and median available gaze-recording duration per question for the gaze-based datasets. VOILA-A is omitted because it uses mouse traces rather than gaze recordings and therefore does not provide directly comparable gaze-timing measurements.

Figure A.1 summarizes the available gaze duration per question in the datasets that contain eye-tracking traces. DriVQA has a mean gaze duration of 4.2 s and a median duration of 4.1 s per question, while VQA-MHUG has a mean duration of 5.8 s and a median duration of 4.8 s. Both durations are short enough that a user asking a 3–6 second question would generate the full trace naturally, without any additional waiting period. The experiment therefore treats the gaze-recording interval as concurrent with the user's spoken ques-

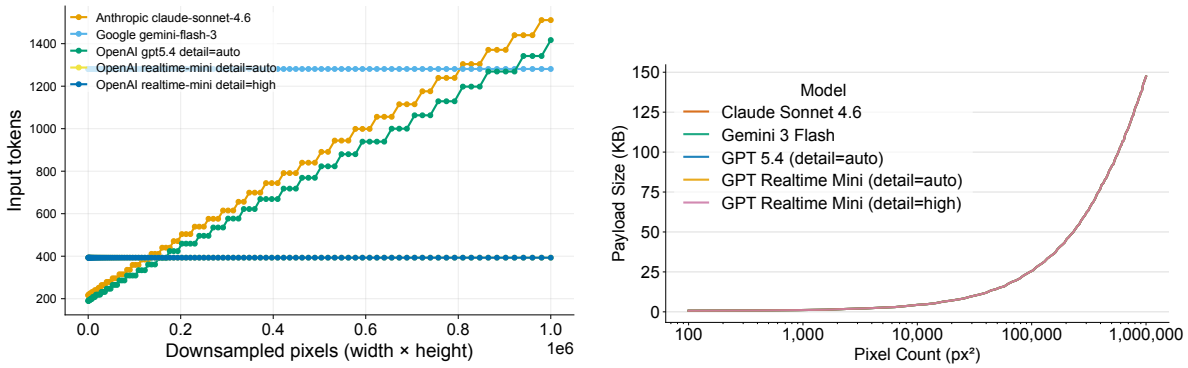
tion rather than as an additional sequential waiting period. Under this assumption, the main extra latency of the gaze-based methods relative to a no-preprocessing baseline is the local gaze-to-ROI preprocessing time, not the full duration of gaze collection.

A live system could implement this concurrency in two ways. One option is continuous gaze buffering, where recent gaze points are stored in a first-in-first-out buffer that retains only the last few seconds of data. When the user submits a question, the system can immediately use the buffered trace corresponding to the utterance window. A second option is voice-triggered collection, where speech activity detection starts accumulating gaze samples as soon as the user begins speaking. Both designs avoid adding the complete gaze-recording window after the question has been asked, although they would still require a real deployment study to validate user behaviour, synchronisation accuracy, and eye-tracker reliability.

A.5. Token sensitivity to image resolution and compression

Two calibrations together identify what drives provider-reported token usage. The resolution calibration shows that both token counts and serialised payload size increase with pixel count, which raises the question of whether it is decoded pixel geometry or encoded file size that determines token accounting. The JPEG compression calibration resolves the ambiguity by varying payload size independently of pixel count.

Resolution calibration. The same image was submitted at 100 square sizes from 10×10 to 1000×1000 pixels with all other settings held fixed. Figure A.2 plots input-token usage (panel a) and serialised payload size (panel b) against pixel count. Input tokens rather than total tokens are shown to avoid confounding visual-input accounting with output-length variation. `gpt-realtime-mini` and `gemini-3-flash-preview` report near-constant token counts across the full tested range, consistent with a fixed or strongly bucketed image-token budget. `gpt-5.4-2026-03-05` and `claude-sonnet-4-6` show stepwise increases consistent with tiled accounting; the first step occurs at the smallest tested resolutions, so neither model exhibits a flat low-resolution region. Serialised payload size grows monotonically with pixel count for all providers. Because token usage and payload size co-vary in this experiment, the resolution calibration alone cannot determine whether it is decoded pixel geometry or encoded byte size that governs token accounting.



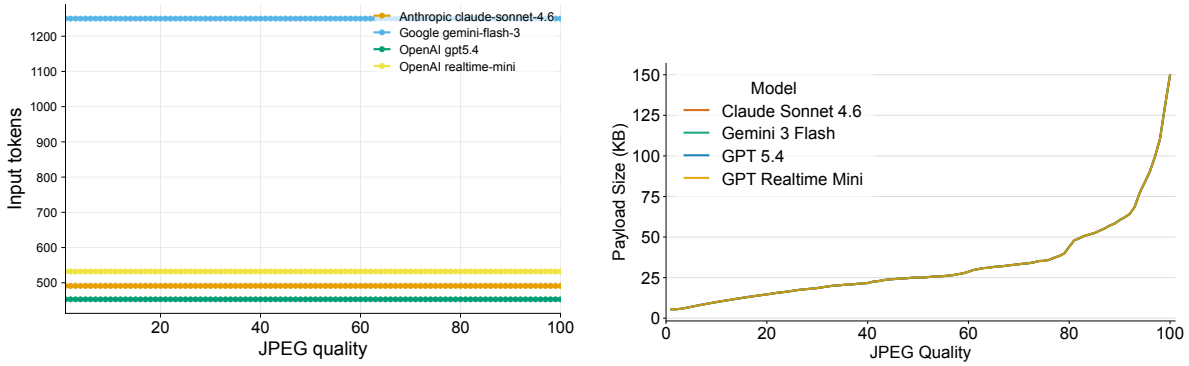
(a) Input Tokens vs Pixel Count

(b) Payload size vs Pixel Count

Figure A.2: Image-resolution scaling calibration. Panel (a) shows provider-reported input token usage against pixel count; panel (b) shows serialised payload size against pixel count. Token counts for `gpt-5.4-2026-03-05` and `claude-sonnet-4-6` increase in steps consistent with tiled accounting, while `gpt-realtime-mini` and `gemini-3-flash-preview` report near-constant counts. Payload size grows monotonically with pixel count for all providers, making the two quantities co-vary in this experiment.

JPEG compression calibration. To break the co-variation between pixel count and payload size, JPEG quality was varied from high to low at fixed 640×327 pixel dimensions. Figure A.3 shows that provider-reported input token usage (panel a) remains constant across the full quality range, while serialised payload size (panel b) decreases substantially at lower quality settings. Payload size therefore varies independently of token usage in this experiment. Taken together, the two calibrations are decisive: when pixel count changes, both

tokens and payload change; when payload size changes through compression at fixed pixel count, tokens do not change. Token accounting is driven by decoded pixel geometry, not by encoded byte size.



(a) Input Tokens vs JPEG Quality

(b) Payload Size vs JPEG Quality

Figure A.3: JPEG compression calibration at fixed 640×327 pixel dimensions. Panel (a) shows provider-reported input token usage across JPEG quality levels; panel (b) shows serialised payload size across the same range. Token counts are constant for all providers while payload size varies substantially, confirming that provider token accounting is driven by decoded pixel geometry and not by encoded file size.

Realtime Mini bucket check. The flat resolution curve for `gpt-realtime-mini` does not mean decoded geometry is irrelevant. Table A.2 compares token counts for square and non-square variants of the same visual content at matched pixel areas. Square images consistently fall into a 194-token bucket, while non-square images of comparable area fall into a 323-token bucket. Padding a 640 × 327 original to 640 × 640 moved it from the high bucket to the low bucket. Changing JPEG quality, image format (PNG/WebP), or the `detail` API parameter did not change bucket assignment. Token accounting for `gpt-realtime-mini` is therefore governed by decoded aspect ratio, not by pixel count, encoded byte size, or API detail setting.

Table A.2: Realtime Mini image-token bucket check for selected decoded image geometries. The prompt and visual content were held constant; only geometry, padding, encoding format, or detail setting was varied.

Comparison	Example dimensions	Pixels	Image tokens	Interpretation
Square resize sweep	100 × 100, 640 × 640, 1000 × 1000	10k–1.0M	194	Pixel count changed by 100×, but square images stayed in the same low bucket.
Padded square	Original image padded to 640 × 640	410k	194	Submitting a square geometry moved the image into the low bucket.
Fixed dimensions, different encoding	JPEG q20/q85/q100, PNG, WebP at 640 × 327	209k	323	Encoded file size and format did not change the bucket.
Similar area, different aspect ratio	457 × 457, 800 × 262, 262 × 800	≈209k	194 or 323	The square variant used 194 tokens; wide and tall variants used 323 tokens.
Detail setting comparison	Same variants with `auto`, `high`, or omitted detail	unchanged	unchanged	The tested detail settings did not change bucket assignment.

A.6. Weighted Trade-off Decision Matrices: Additional Providers

The same weighted trade-off analysis from subsection 6.2.1 was run on the three REST providers in the cross-provider experiment. Scenario weights from Table 6.1 were applied to each provider independently, and scores were normalized within each provider's method set so that the best technique for each metric receives a score of 1 and the worst receives 0. Ranks and scores are not comparable across providers; each matrix shows the relative ordering within that provider's evaluated methods.

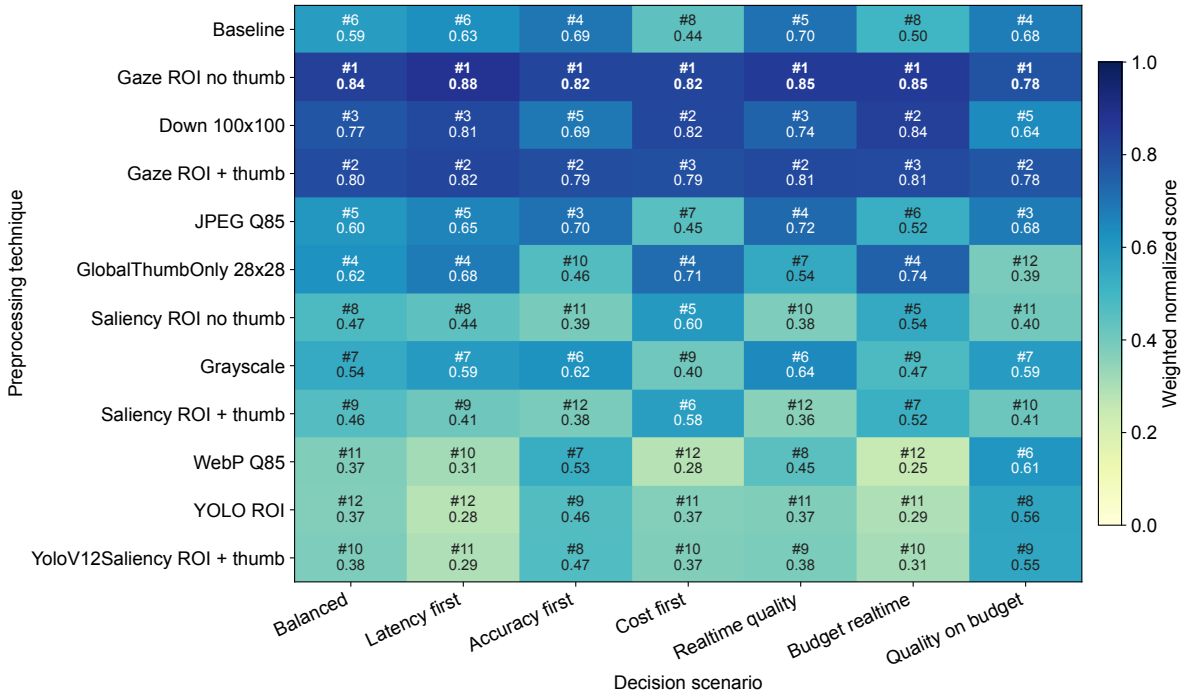


Figure A.4: Weighted trade-off decision matrix by preprocessing technique for the `gpt-5.4` condition. `GazeRoi_NoGlobalThumb` ranks first across all seven decision scenarios, consistent with the large payload and cost reductions that gaze ROI achieves on a high-cost model tier.

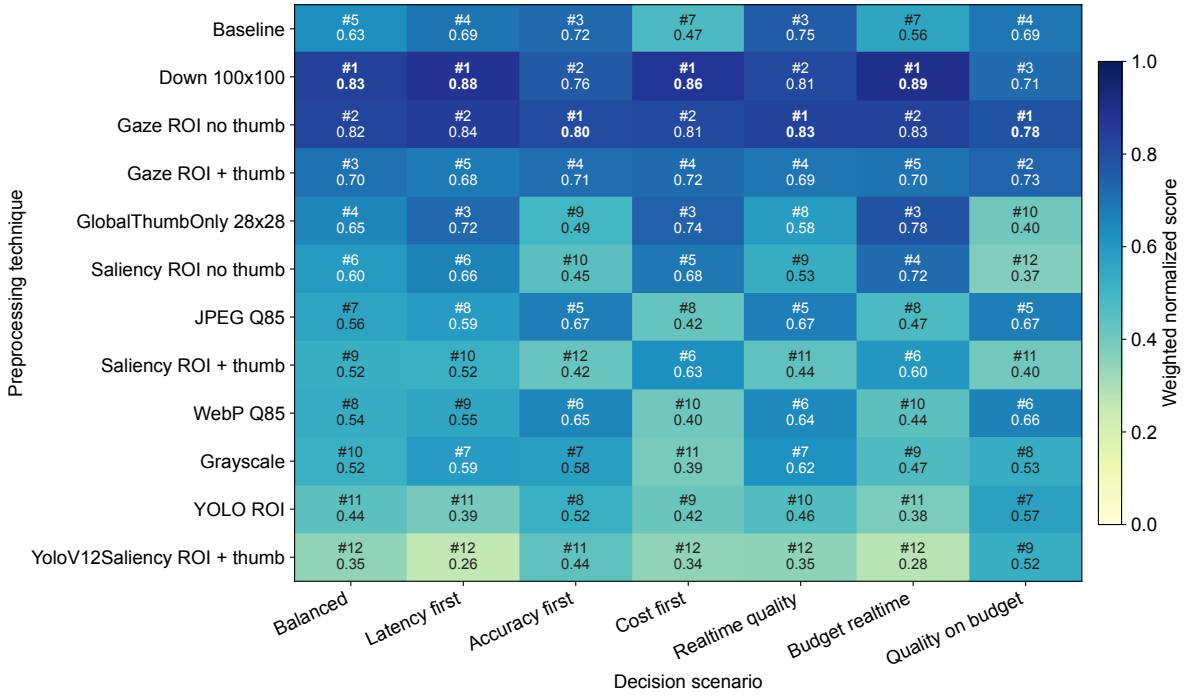


Figure A.5: Weighted trade-off decision matrix by preprocessing technique for the `claude-sonnet-4-6` condition. `Downsampler_100x100` ranks first in six of seven scenarios; `GazeRoi_NoGlobalThumb` ranks first only in the accuracy-first scenario.

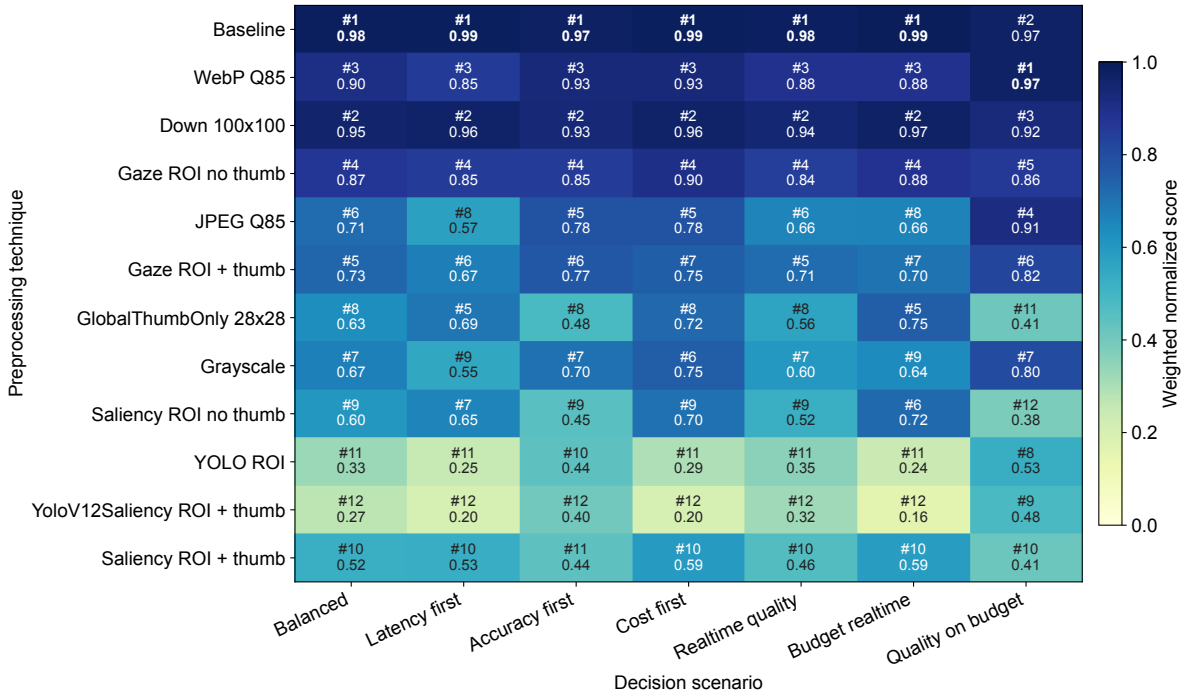


Figure A.6: Weighted trade-off decision matrix by preprocessing technique for the `gemini-3-flash-preview` condition. The no-preprocessing baseline ranks first in six of seven scenarios because most preprocessing techniques increased total pipeline latency on this provider under the REST recording conditions.

A.7. JPEG Quality Versus Latency Sensitivity

To separate the effect of encoded byte size from decoded visual geometry, total pipeline latency was measured across the full JPEG quality range (quality 1–95) at fixed pixel dimensions (640 × 327). Figure A.7 through Figure A.10 plot latency against JPEG quality for each of the four evaluated models, with a linear trend line and coefficient of determination (R^2) shown.

The results split clearly along interface type. `gpt-realtime-mini` is the only model sensitive to JPEG quality ($R^2 = 0.10$): larger payloads at higher quality add bytes to the streaming connection, which slightly increases pipeline latency. The three REST-based models — `gpt-5.4-2026-03-05` ($R^2 = 0.02$), `gemini-3-flash-preview` ($R^2 = 0.00$), and `claude-sonnet-4-6` ($R^2 = 0.00$) — show flat relationships. For these providers, server-side compute time dominates; JPEG compression reduces transmission overhead but does not change how long the model takes to respond.

Figure A.7: JPEG quality versus total pipeline latency for `gpt-realtime-mini` ($R^2 = 0.10$). A weak positive trend is visible: larger payloads at higher quality levels slightly increase latency on the streaming connection.

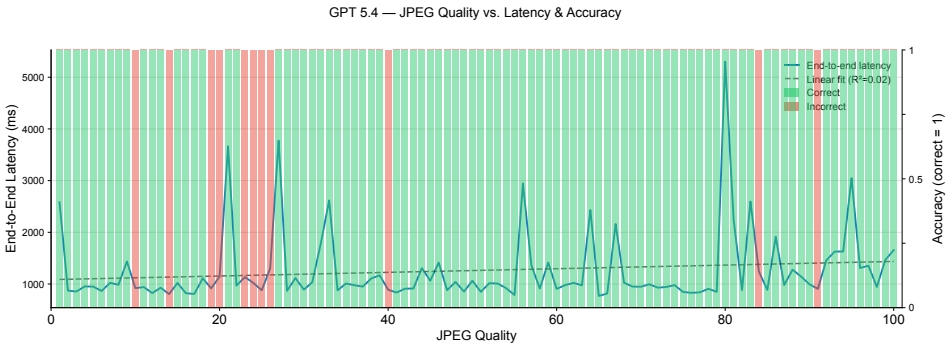


Figure A.8: JPEG quality versus total pipeline latency for `gpt-5.4-2026-03-05` ($R^2 = 0.02$). The trend is effectively flat. Outlier latency spikes around quality 20–25 inflate variance without producing a systematic payload-latency relationship.

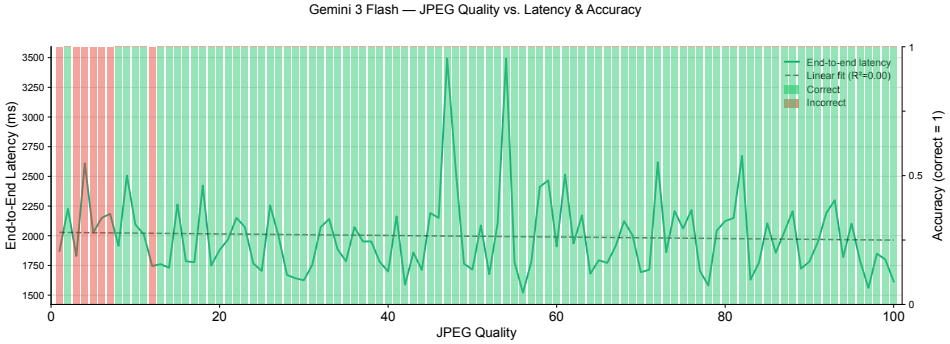


Figure A.9: JPEG quality versus total pipeline latency for `gemini-3-flash-preview` ($R^2 = 0.00$). Latency is independent of JPEG quality, with a stable baseline near 2 000 ms across the full quality range.

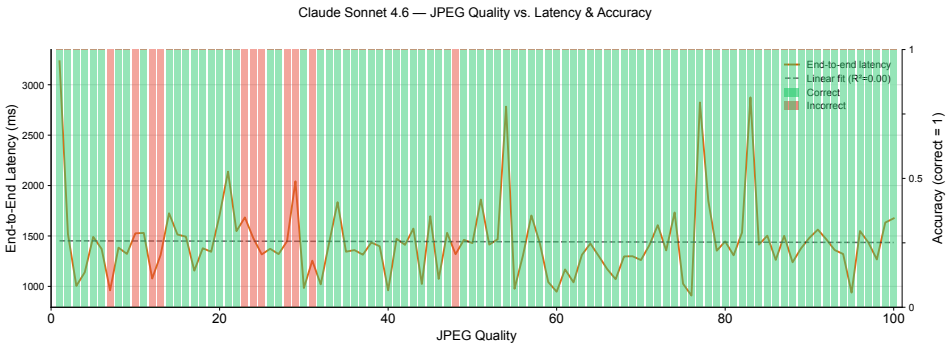


Figure A.10: JPEG quality versus total pipeline latency for `claude-sonnet-4-6` ($R^2 = 0.00$). The trend is completely flat, confirming that JPEG compression quality has no measurable effect on pipeline latency for this REST provider.

A.8. Claude and Gemini REST Results

Tables A.3 and A.4 give the full latency decompositions for the two REST providers not covered in the main results chapter. For `gemini-3-flash-preview`, every preprocessing technique increases total pipeline latency relative to the no-preprocessing baseline, with increases ranging from +2% to +119%. For `claude-sonnet-4-6`, the three lightest techniques (`Downsampler`, `GlobalThumbOnly`, `GazeROI`) reduce total pipeline latency by 2–5%, while heavier techniques and those that add a global thumbnail increase latency.

Table A.3: Latency decomposition and baseline-relative change for the `gemini-3-flash-preview` model (REST API). All preprocessing techniques increase total pipeline latency relative to the baseline, with increases ranging from +2% to +119%.

	Local pre-processing		Request sent → First response		First response → Done †		Total pipeline	
Technique	ms	Δ%	ms	Δ%	ms	Δ%	ms	Δ%
Baseline	2.3	0	1792.4	0	2.2	0	1796	0
Downsampler	17.7	**+671**	1805.9	+1	2.2	+1	1825	+2
GlobalThumbOnly	16.9	**+639**	2067.8	**+15**	2.3	+5	2087	**+16**
SalientROI	41.9	**+1728**	2167.4	**+21**	2.2	−2	2211	**+23**
GazeROI	15.6	**+581**	2205.1	**+23**	2.3	+6	2223	**+24**
WebP Q85	91.6	**+3895**	2326.1	**+30**	2.4	+8	2420	**+35**
SalientROI+T	48.3	**+2006**	2688.7	**+50**	2.9	**+31**	2739	**+52**
GazeROI+T	26.5	**+1055**	2858.4	**+59**	2.2	+2	2887	**+61**
Grayscale	13.9	**+508**	3507.1	**+96**	2.3	+6	3523	**+96**
JPEG Q85	13.3	**+481**	3611.9	**+102**	2.2	+1	3627	**+102**
YOLOv12ROI	86.8	**+3684**	3829.3	**+114**	2.2	−1	3918	**+118**
YOLOv12ROI+T	93.2	**+3964**	3844.9	**+115**	2.4	+9	3940	**+119**

† `gemini-3-flash-preview` returns responses in a single chunk (no token streaming); *First response → Done* is ≤3 ms throughout.

Table A.4: Latency decomposition and baseline-relative change for the `claude-sonnet-4-6` model (REST API). `Downsampler`, `GlobalThumbOnly`, and `GazeROI` reduce total pipeline latency by 2–5% relative to baseline; techniques adding a global thumbnail or using YOLO increase latency.

	Local pre-processing		API call time †		Total pipeline	
Technique	ms	Δ%	ms	Δ%	ms	Δ%
Downsampler	7.0	**+227**	1546.2	−5	1553	−5
GlobalThumbOnly	4.8	**+126**	1589.6	−3	1594	−2
GazeROI	9.0	**+322**	1597.0	−2	1606	−2
Baseline	2.1	0	1630.7	0	1633	0
SalientROI	38.4	**+1697**	1624.4	+0	1663	+2
Grayscale	13.8	**+543**	1675.7	+3	1689	+3
JPEG Q85	13.4	**+527**	1764.4	+8	1778	+9
GazeROI+T	21.8	**+918**	1799.7	**+10**	1821	**+12**
WebP Q85	88.0	**+4018**	1740.5	+7	1829	**+12**
SalientROI+T	45.9	**+2047**	1846.1	**+13**	1892	**+16**
YOLOv12ROI	85.6	**+3902**	1990.3	**+22**	2076	**+27**
YOLOv12ROI+T	90.6	**+4136**	2151.5	**+32**	2242	**+37**

† TTFT and token-generation phase were not captured for `claude-sonnet-4-6`; full end-to-end API call time is reported instead.