software

MDPI

Article

UVLM: A Modular Python Package for Unified Vision–Language Model Loading, Inference and Comparison

Joan Perez [1,*] and Giovanni Fusco [2]

[1] Urban Geo Analytics, 13001 Marseille, France
[2] ESPACE, Université Côte d'Azur–CNRS–AMU–Avignon Université, 06200 Nice, France; giovanni.fusco@univ-cotedazur.fr
* Correspondence: jperez@urbangeoanalytics.com

Abstract

Vision–Language Models (VLMs) have emerged as powerful tools for image understanding tasks, yet their practical deployment remains hindered by significant architectural heterogeneity across model families. This paper introduces UVLM (Unified Vision–Language Model), a pip-installable Python (v3.9+) package that provides a unified interface for loading, configuring, and running multiple VLM architectures on custom image analysis tasks. UVLM currently supports two major model families which differ fundamentally in their vision encoding, tokenization, and decoding strategies: LLaVA-NeXT and Qwen2.5-VL. The package abstracts these differences behind a single inference function and eliminates all architecture-specific code from the user's workflow. UVLM is organized as eight modular Python components (model loading, dual-backend inference, response parsing, consensus validation, batch processing, prompt assembly, model registry, and utilities) and can be deployed in three modes: Google Colab for zero-install cloud access, local Jupyter notebooks for on-premises GPU use, and as a programmatic API for integration into automated pipelines. Key features include a multi-task prompt builder supporting four response types (numeric, category, boolean, text), a consensus validation mechanism based on majority voting, a flexible token budget (up to 1500 tokens) for custom reasoning strategies, and built-in truncation detection. The package is designed for extensibility: adding a new VLM family requires implementing one backend-specific inference section and adding entries to the model registry, without modifying any other module. An illustrative example on 120 street-view images across 16 model configurations is provided to demonstrate the software's evaluation workflow.

Keywords: Vision–Language Models; Python package; unified inference; LLaVA; Qwen; multimodal; chain-of-thought; consensus validation

Academic Editor: Ian Gorton

Received: 12 May 2026
Revised: 2 July 2026
Accepted: 8 July 2026
Published: 9 July 2026

1. Motivation and Significance

Vision–Language Models (VLMs) represent a class of multimodal architectures that combine a visual encoder with a large language model decoder, enabling systems to interpret images and respond to natural language queries implying reasoning about their content. In recent years, the field has witnessed a rapid proliferation of VLM architectures, with prominent families including LLaVA [1], Qwen-VL [2], BLIP-2 [3], and InternVL [4], among others. Each family introduces its own design choices regarding vision encoding, cross-modal alignment, tokenization strategies, and decoding protocols. While this

diversity has accelerated progress in multimodal AI, it has also created a significant practical challenge: researchers who wish to compare models across families must write and maintain separate inference pipelines for each architecture, even when the evaluation task is identical.

This fragmentation is particularly problematic for applied researchers outside the core machine learning community. In fields such as urban studies, remote sensing, medical imaging, or environmental monitoring, practitioners increasingly seek to leverage VLMs as zero-shot visual analysis tools. However, deploying even a single model requires understanding architecture-specific processor classes, chat template formats, image preprocessing pipelines, and generation configurations. Deploying multiple models for systematic comparison multiplies this complexity. As a result, many applied studies rely on a single model without evaluating alternatives and possibly using suboptimal solutions, limiting the robustness and generalizability of their findings.

The two VLM families currently supported by UVLM (LLaVA-NeXT [1,5] and Qwen2.5-VL [2]) illustrate the nature of this heterogeneity. LLaVA adopts a comparatively simple architecture: a CLIP-based vision encoder [6] connected to a large language model through a trained MLP projection layer. The inference pipeline relies on a dedicated `LlavaNextProcessor` class that handles image-text interleaving, and the raw output is decoded directly from the full generated sequence, requiring post-processing to strip prompt tokens and instruction markers. Qwen2.5-VL, by contrast, employs a redesigned Vision Transformer (ViT) with RMSNorm, SwiGLU activations, and window-based attention, supporting native-resolution image inputs. Its inference pipeline uses a generic `AutoProcessor` with separate vision processing utilities (`process_vision_info`) and requires explicit trimming of input token IDs from the generated output before decoding. These differences extend to memory management (Qwen models require additional visual token budget parameters; min/max pixel constraints) and to generation configuration, where Qwen loads settings from the model config object rather than accepting them as direct arguments. These are not superficial API variations; they reflect fundamentally different transformer logic in how visual information is encoded, merged with text tokens, and decoded into language.

UVLM was developed to address this challenge by providing a single, unified interface that abstracts away architecture-specific details while preserving full control over inference parameters. The tool originated in the context of the SAGAI project (Streetscape Analysis with Generative Artificial Intelligence) [7], a workflow for scoring street-level urban scenes using VLMs and open-access geospatial data, and the emc2 project [8], a European initiative studying the 15 min city model in suburban areas, which has also produced open-source geospatial tools such as PPCA [9]. Within these projects, researchers needed to systematically evaluate how different VLMs perform on identical visual analysis tasks using exactly the same prompts and evaluation protocols. Writing and debugging separate pipelines for each model was both error-prone and time-consuming, motivating the creation of a unified loader.

Existing tools address different layers of the VLM stack. Hugging Face Transformers provides the model loading and generation primitives but requires architecture-specific code for each family (processor classes, chat templates, token trimming). Inference servers (vLLM, SGLang) optimise model execution and serving performance for production deployments, while orchestration frameworks (LangChain) provide abstractions for integrating models into application workflows and multi-component pipelines. Evaluation suites such as VLMEvalKit [10] target standard academic benchmarks with fixed datasets and metrics. UVLM occupies a different position: it is a unified loader, structured response parser, consensus validator, and batch evaluation engine built on top of Hugging Face

Transformers, designed for applied researchers who need to compare open-weight VLMs on custom image analysis tasks with their own prompts, images, and evaluation protocols.

UVLM is distributed as a pip-installable Python package (v3.0.2, Python $\geq$3.9) that can be installed from GitHub (`pip install git+` https://github.com/perezjoan/UVLM.git, accessed on 25 June 2026). The package metadata, dependency versions, and build configuration are specified in `pyproject.toml`; core dependencies include PyTorch (>=2.0), Transformers (>=4.47), Accelerate, BitsAndBytes, and `qwen-vl-utils`. PyTorch with CUDA support must be installed separately to match the user's GPU driver. UVLM can be deployed in three modes: Google Colab notebooks for zero-install cloud access, local Jupyter notebooks for on-premises GPU use, and as a programmatic Python API for integration into automated pipelines. It leverages the Hugging Face Transformers library [11] and the BitsAndBytes quantization library [12] to support models ranging from 3B to 110B parameters across multiple precision modes (FP16, 8-bit, 4-bit quantization). The framework is designed for extensibility, and future releases will progressively integrate additional VLM families (e.g., InternVL [4], BLIP-2 [3], CogVLM) as their adoption grows in the research community.

The UVLM source code is publicly available under the Apache 2.0 license at https://github.com/perezjoan/UVLM (accessed on 25 June 2026), with tagged releases, version history, and a permanent archive on Zenodo (DOI: https://zenodo.org/records/20917845). The repository includes a minimal reproducible example that downloads a sample image and runs inference in five lines of Python.

2. Software Description

2.1. Package Architecture

UVLM is organized as a modular Python package comprising eight modules, each with a single responsibility (Figure 1). This design separates core inference logic from user interface code: the package provides a programmatic API, while interactive notebook interfaces serve as thin wrappers that import from the package.

The eight modules are: `loader.py` (model loading with quantization and device placement), `inference.py` (dual-backend forward pass for LLaVA and Qwen), `parsers.py` (type-specific response parsing), `consensus.py` (multi-run majority voting), `batch.py` (folder-level batch processing with CSV output), `prompts.py` (prompt assembly and chain-of-thought templates), `registry.py` (model checkpoint registry), and `utils.py` (seed management, environment detection, HuggingFace token retrieval).

A key design decision is the elimination of all global state. The `load_model()` function returns a `model_ctx` dictionary containing all model-related state (model, processor, backend type, device information), which is then passed explicitly to every subsequent function. This pattern makes the code testable, allows multiple models to be loaded in the same session, and eliminates hidden coupling between modules.

2.2. Deployment Modes

UVLM supports three deployment modes, all sharing the same underlying package code and producing identical outputs.

Google Colab. The Colab notebook (`notebooks/UVLM_colab.ipynb`) installs the package automatically via `pip install` in the first cell and mounts Google Drive for image access. The HuggingFace authentication token is retrieved from Colab's built-in secrets manager. This mode requires no local hardware since a free Google account with a T4 GPU runtime is sufficient for models up to 7B parameters with 4-bit quantization.

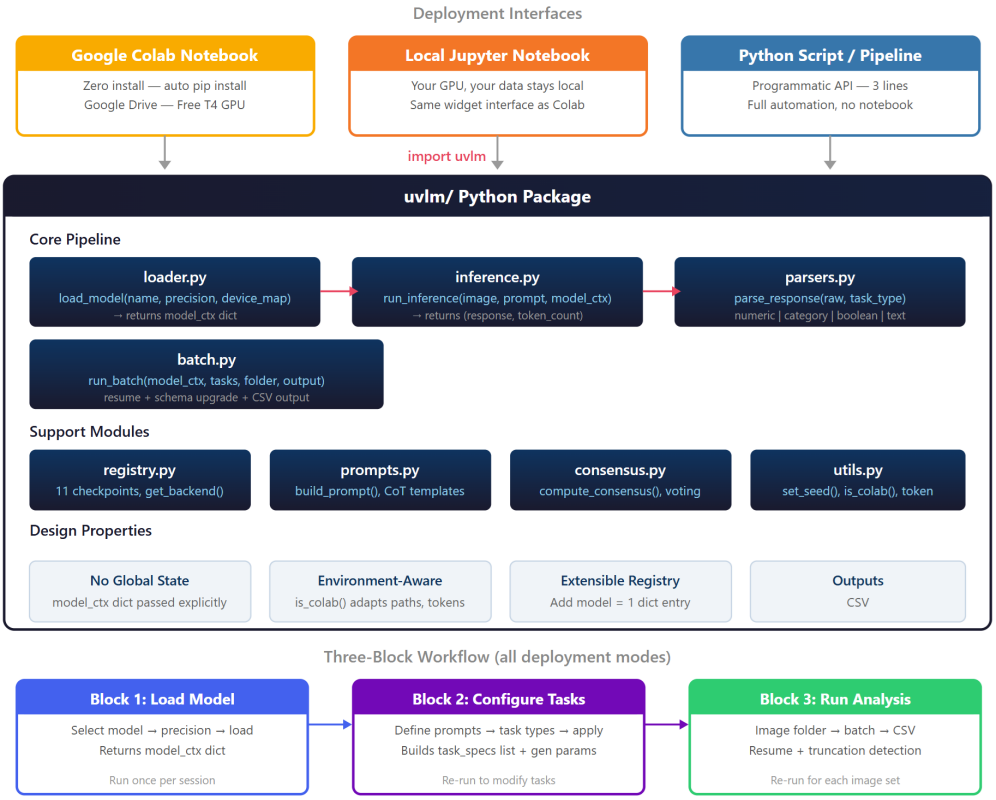

Figure 1. UVLM software architecture. The package comprises eight Python modules (**center**) providing the core API. Three deployment interfaces (**top**) import from the package. The three-block workflow (**bottom**) is preserved across all deployment modes.

Local Jupyter Notebook. The local notebook (`notebooks/UVLM_local.ipynb`) provides the same widget-based interface but reads images from local folders. The HuggingFace token is retrieved from the `HF_TOKEN` environment variable or the `huggingface-cli` login cache. This mode is suitable for researchers with their own GPU hardware, for processing sensitive data that cannot be uploaded to cloud services, or for environments where network latency to HuggingFace model servers is a concern.

Python Script. The package exposes a clean programmatic API that can be used without any notebook. Only three lines of code are required to load a model, run inference on an image, and parse the result:

```
from uvlm import load_model, run_inference, parse_response
ctx = load_model("[Qwen] Qwen2.5-VL 7B Instruct", precision="4bit")
raw, tokens = run_inference("photo.jpg", "Count the cars", ctx)
result = parse_response(raw, "numeric")
```

For batch processing, the `run_batch()` function handles the full pipeline from image folder to CSV output, including resume mode and schema upgrading.

Environment detection is handled automatically: the `is_colab()` utility in `utils.py` detects the runtime environment and adapts default paths and token retrieval accordingly, so the same package code works in all three modes without user intervention.

2.3. Three-Block Workflow

Regardless of deployment mode, UVLM follows a three-block workflow. Block 1 (Model Loading) loads the selected VLM and configures hardware settings. The model registry currently includes seven LLaVA-NeXT variants (Mistral 7B, Vicuna 7B/13B, 34B, LLaMA3 8B, 72B, 110B) and four Qwen2.5-VL variants (3B, 7B, 32B, 72B Instruct). Users

configure the precision mode (FP16, 8-bit, or 4-bit quantization via BitsAndBytes [12]) and the device placement strategy (single GPU, automatic multi-device, or CPU offload). For Qwen models, additional visual token budget parameters (minimum and maximum pixel constraints) are configured to manage memory usage.

Block 2 (Task Configuration) defines up to ten analysis tasks through an interactive form. For each task, the user specifies a column name, a text prompt, and a task type (numeric, category, boolean, or text). Each task can optionally enable consensus validation and advanced reasoning. Global generation parameters include temperature, top-p sampling, maximum token count (adjustable up to 1500 tokens), and an optional fixed random seed.

Block 3 (Batch Execution) iterates over all images in a user-specified folder, executes all configured tasks sequentially for each image, and writes results to CSV. The engine supports resume mode (detecting already-processed images), schema upgrading (appending missing columns when new tasks are added between runs), per-task error handling, periodic checkpoint saves, and truncation detection.

2.4. Dual-Backend Inference

The central technical contribution of UVLM is its unified inference abstraction, implemented in `inference.py`, which routes each call to the appropriate backend pipeline based on the `model_ctx["backend"]` value. The `run_inference()` function accepts an image path, a prompt, and the model context dictionary, and returns a tuple `(raw_response, token_count)`. The two pipelines are illustrated in Figure 2.

LLaVA-NeXT pipeline. The LLaVA backend constructs a conversation object following the chat template format, with the image embedded as a typed content block alongside the text prompt. The `LlavaNextProcessor` applies the chat template, tokenizes the combined input, and returns tensors ready for generation. After the forward pass, the full output sequence including the echoed prompt is decoded, and the response is extracted by stripping architecture-specific markers (e.g., `[/INST]`, `ASSISTANT:` patterns) that vary depending on the base LLM (Mistral, Vicuna, LLaMA3).

Qwen2.5-VL pipeline. The Qwen backend constructs a message list with explicit image and text content types. Before tokenization, a dedicated `process_vision_info()` function extracts and preprocesses the visual inputs separately. The `AutoProcessor` then tokenizes text and visual features jointly. Generation uses a `GenerationConfig` object loaded from the model configuration. Crucially, the Qwen pipeline requires an explicit token-trimming step after generation: the prompt token IDs are subtracted from the output sequence before decoding, as the model returns the full concatenated sequence. This contrasts with the LLaVA approach, where prompt removal is performed on the decoded text via string matching.

Both pipelines converge at the `parse_response()` function in `parsers.py`, which applies task-type-specific parsing to the cleaned text output. For numeric tasks, the parser extracts the last number found in the response via regular expression. For category tasks, it strips common response prefixes and returns the cleaned label. For boolean tasks, it normalizes variations (yes/no/true/false) to a binary value. For text tasks, the raw response is returned as-is. When parsing fails, the value "NA" is recorded. Since the inference path calls `model.generate()` directly and all parsing operations are string-level, the overhead introduced by the UVLM wrapper is negligible relative to the GPU forward pass.

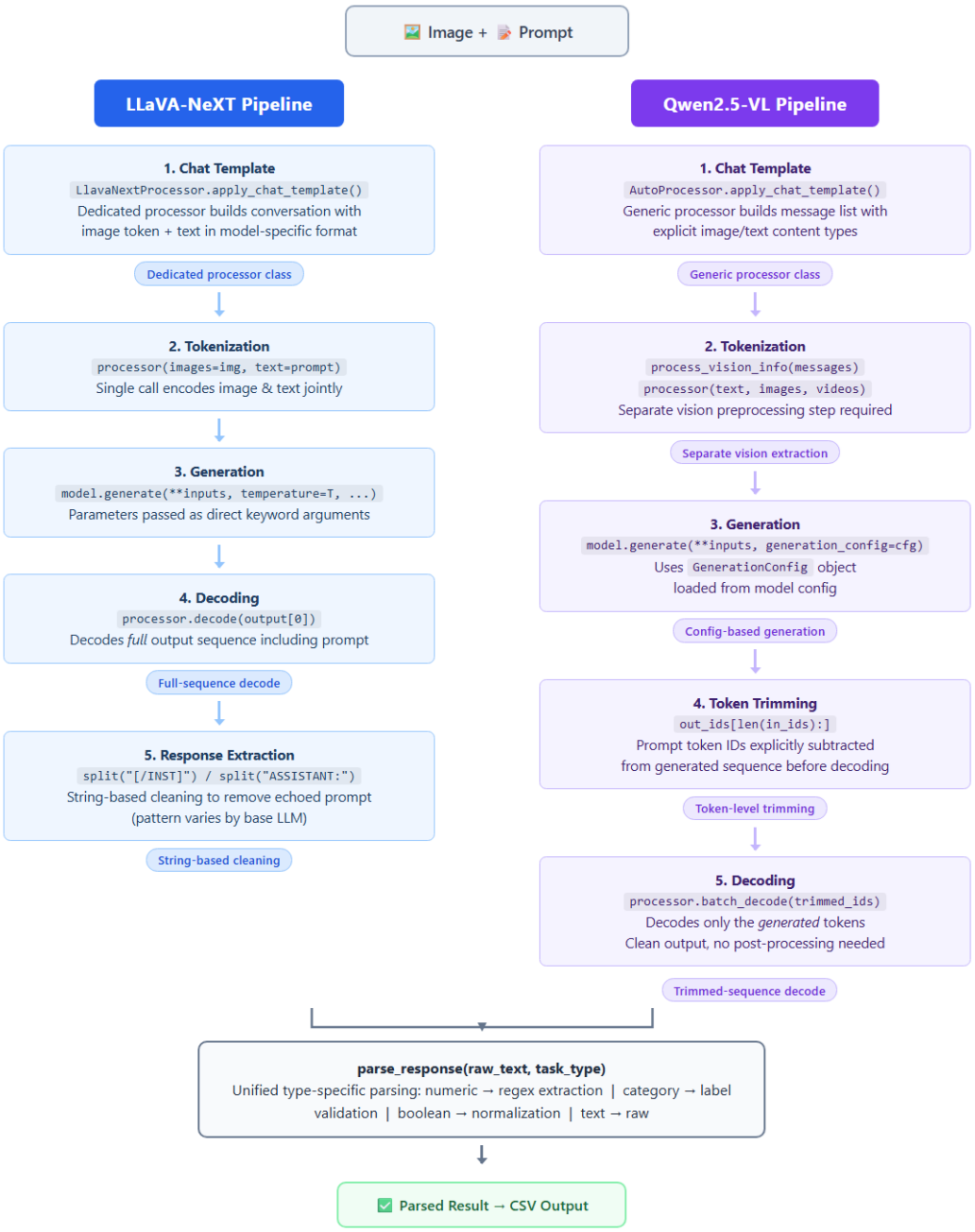

Figure 2. Comparative diagram of the dual-backend inference pipelines in UVLM. Both pipelines receive the same image–prompt pair and converge at the unified response parser. The ** notation denotes Python dictionary unpacking.

2.5. Prompt Engineering and Reasoning Support

UVLM structures each task prompt as a concatenation of four elements: a role instruction (shared across all tasks, defining the model's persona), a task description (the specific question), a theory section (definitions, rules, and edge cases relevant to the task), and a format specification (expected output structure). The `build_prompt()` function in `prompts.py` assembles these four components. UVLM supports only single-task prompts, where each prompt requests a single piece of information (e.g., numeric, categorical, Boolean, or free-text). Multi-task prompts are not handled correctly and should instead be decomposed into multiple single-task prompts. This modular prompt architecture allows users to control prompt complexity independently of the underlying model, facilitating systematic experiments on prompt sensitivity.

In standard mode (50 max tokens), model outputs are typically short and contain only the requested value, making the extraction rules reliable. Users can implement custom reasoning strategies by writing task prompts that request step-by-step explanations and increasing the maximum token budget (up to 1500 tokens) to accommodate longer outputs. As a convenience for benchmarking, UVLM also provides a built-in advanced reasoning reference mode. When enabled, the format instruction is automatically replaced with a standardized chain-of-thought (CoT) directive [13] that asks the model to first describe its observations, then reason step by step, and finally provide a formatted answer (e.g., `ANSWER: <integer>` for numeric tasks). The token budget is automatically set to 1024 tokens. The response parser operates in two tiers: it first scans the last five lines of the output for the `ANSWER:` pattern, and falls back to the standard last-number extraction if the pattern is absent. This ensures that intermediate numbers produced during reasoning do not override the final answer, while providing graceful degradation when the model omits the expected format. The reasoning trace is stored in a dedicated CSV column for inspection.

2.6. Consensus Validation

VLM outputs are inherently stochastic when temperature sampling is used. To improve reliability, UVLM implements a consensus validation mechanism inspired by the self-consistency principle [14]: each task is executed multiple times on the same image, and the final answer is determined by majority voting across all runs. Consensus is enabled per task with a configurable number of runs (2 to 5, default: 2). The `compute_consensus()` function in `consensus.py` filters out NA results (parsing failures) before voting, ensuring that failed runs do not influence the final answer, while the agreement ratio is computed over all runs (including failures) to preserve the reliability metric. For numeric tasks, an optional tolerance parameter allows grouping values within a specified percentage of each other.

2.7. Supported Models

Table 1 lists all model checkpoints currently supported by UVLM. The internal registry in `registry.py` is structured as a Python dictionary mapping display names to (backend, checkpoint_id) tuples, facilitating both maintenance and expansion.

Table 1. Supported VLM checkpoints in UVLM.

Family	Model	Params	Checkpoint ID
LLaVA-NeXT	Mistral 7B	7B	`llava-hf/llava-v1.6-mistral-7b-hf`
	Vicuna 7B	7B	`llava-hf/llava-v1.6-vicuna-7b-hf`
	Vicuna 13B	13B	`llava-hf/llava-v1.6-vicuna-13b-hf`
	34B	34B	`llava-hf/llava-v1.6-34b-hf`
	LLaMA3 8B	8B	`llava-hf/llama3-llava-next-8b-hf`
	72B	72B	`llava-hf/llava-next-72b-hf`
	110B	110B	`llava-hf/llava-next-110b-hf`
Qwen2.5-VL	3B Instruct	3B	`Qwen/Qwen2.5-VL-3B-Instruct`
	7B Instruct	7B	`Qwen/Qwen2.5-VL-7B-Instruct`
	32B Instruct	32B	`Qwen/Qwen2.5-VL-32B-Instruct`
	72B Instruct	72B	`Qwen/Qwen2.5-VL-72B-Instruct`

Models with 72B+ parameters exceed single-GPU memory even with 4-bit quantization and require multi-GPU environments. In practice, models up to 34B parameters can be loaded on a single GPU (T4 or A100) using 4-bit quantization, which covers the majority of the supported registry.

2.8. Extensibility

UVLM is designed so that adding support for a new VLM family is localised to a small number of modules, without modifying the parsing, consensus, batch processing, or prompt logic.

Step 1: Registry. A new model family is registered by adding entries to the `MODEL_CHOICES` dictionary in `registry.py`. Each entry maps a display name to a tuple of (backend_name, huggingface_checkpoint_id). For example, adding an InternVL model would require one line: `"[InternVL] InternVL2-8B": ("internvl", "OpenGVLab/InternVL2-8B")`.

Step 2: Inference backend. The `run_inference()` function in `inference.py` dispatches on the backend name. Adding a new family requires implementing the backend-specific sections: (a) how to construct the input (processor class, chat template, vision preprocessing), (b) how to call `model.generate()` (generation config format), and (c) how to extract the response from the output (token trimming or string cleaning). The model loading logic in `loader.py` similarly requires a new branch for the processor and model class imports.

All other modules (parsers, consensus, batch processing, prompts, utilities) are backend-agnostic and require no changes. The effort required to integrate a new family depends on how much that family's inference pipeline diverges from existing backends. The two families currently supported (LLaVA-NeXT and Qwen2.5-VL) differ substantially in vision encoding, tokenisation, and decoding, and each required a dedicated backend implementation. A family with similar conventions to an existing backend would require less effort, while an architecturally novel family may require more. Integration of additional families is in progress.

3. Illustrative Example

To demonstrate UVLM's capabilities, we present a comparative evaluation of eight model checkpoints on a dataset of 120 street-level photographs.

3.1. Dataset and Protocol

The dataset comprises 120 street-level images captured in the Nice–Côte d'Azur and Lille-Roubaix-Tourcoing metropolitan areas (France), depicting diverse street frontages [15]. The purpose of this evaluation is to demonstrate UVLM's workflow and its ability to structure cross-architecture comparisons, not to establish definitive model rankings. A single expert annotator scored all images on five tasks of increasing reasoning complexity: (1) motor vehicle counting, (2) sidewalk detection, (3) pedestrian entrance counting, (4) street frontage length estimation in meters, and (5) vegetation type classification (6-class ordinal scheme) (Figure 3). The full prompt specifications are provided in Appendix A and both the human annotations and the complete per-task metrics are provided in Appendix B.

We tested eight model checkpoints in two inference modes (standard and advanced reasoning), yielding 16 configurations. Standard mode used 50 max tokens with 3-run consensus; reasoning mode used 1024 max tokens with 3-run consensus. All models used greedy decoding with a fixed random seed. The same prompt was used for all models without per-model optimisation; the results therefore reflect one prompt configuration and should not be interpreted as each model's performance ceiling.

Visual input example, S27.jpg

User: Task 1—motor vehicle counting (integer)
Human annotation: 3
Qwen2.5-VL-32B_standard: 3
Qwen2.5-VL-32B_reasoning: There are three motor vehicles visible in the foreground of the image. The first vehicle is a silver car parked closest to the camera. [. . .] - Silver car: 1 - Blue car: 1 - Partially hidden car: 1 ### Answer: 3

User: Task 4—street frontage length (continuous, meters)
Human annotation: 15
Qwen2.5-VL-32B_standard: 20
Qwen2.5-VL-32B_reasoning: [. . .] Car 1: 4 m. Space between cars: 4 m. Car 2: 4 m. Additional space beyond Car 2: 4 m. Total estimated length = 4 + 4 + 4 + 4 = 16 m. [. . .] Final Answer: 16

Figure 3. Qwen2.5-VL-32B outputs (standard and reasoning) as compared to human annotations. In standard mode, the model outputs a single value; in reasoning mode, it generates an explicit chain-of-thought before the final answer.

3.2. Results

Model performance was evaluated using a proximity score that rewards close answers rather than requiring exact matches (Equation (1)).

$$\text{Proximity} = \frac{1}{N}\sum_{i=1}^{N}\max\left(0,\ 1-\frac{|y_i-\hat{y}_i|}{R}\right) \quad (1)$$

where y_i is the human value, $\hat{y}_i$ is the model prediction, and R is the observed range of human values. The overall proximity is computed as the unweighted mean across all five tasks. This simple aggregation is not intended as a definitive ranking criterion because the five tasks differ in nature and difficulty, and weighting them equally is one choice among others. For this reason, per-task proximity scores are reported separately in Table 2, allowing readers to assess model behaviour on each task independently of the aggregation method.

The results (Table 2) illustrate the type of cross-architecture patterns that UVLM is designed to reveal.

Model size does not predict performance. In this sample, LLaVA Vicuna 7B in standard mode (83.1%) scored higher than all LLaVA models up to 34B, while LLaVA 34B scored lowest (62.2%), with particularly poor sidewalk detection (15.8% proximity score). This illustrates the value of systematic model comparison for task-specific selection.

Reasoning helps selectively. Advanced reasoning improved Qwen 32B by +4.7 points overall, driven by a +27.1 point gain on length estimation. For most LLaVA models, reasoning degraded performance, primarily because verbose outputs introduced parsing failures (NA rates up to 10.9% for LLaMA3 8B reasoning vs. 0.3% standard).

Table 2. Model comparison by proximity score (%). † Reduced sample due to CUDA crashes. ‡ Partial benchmark due to prohibitive computation time.

#	Model	N	T1 Veh.	T2 Side.	T3 Entr.	T4 Len.	T5 Veg.	Mean
1	Qwen 32B reas	120	97.7	85.0	90.4	82.5	84.2	**88.0**
2	Qwen 32B std	120	98.1	80.8	92.3	55.4	89.8	83.3
3	LLaVA Vic7B std	120	95.4	84.2	85.7	76.5	74.0	83.1
4	LLaVA Vic13B std	120	96.1	80.8	86.2	56.0	70.3	77.9
5	Qwen 7B reas	120	95.8	62.5	84.9	73.3	70.7	77.4
6	LLaVA Vic13B reas	120	93.6	65.8	80.7	70.4	74.5	77.0
7	LLaVA Vic7B reas	120	93.0	77.5	80.9	67.1	66.3	77.0
8	Qwen 3B reas	118 †	96.1	56.8	85.5	66.5	75.4	76.1
9	Qwen 3B std	60 †	96.5	75.0	77.5	70.1	60.7	76.0
10	Qwen 7B std	120	97.5	40.0	89.0	70.1	80.7	75.4
11	LLaVA Mis7B std	120	95.8	45.0	85.7	75.4	70.3	74.4
12	LLaVA 34B reas	25 ‡	88.0	80.0	76.5	70.4	56.0	74.2
13	LLaVA Mis7B reas	120	94.0	70.0	83.9	65.9	52.8	73.3
14	LLaMA3 8B reas	120	93.7	46.7	84.9	66.3	65.3	71.4
15	LLaMA3 8B std	120	95.9	51.7	86.0	32.8	60.3	65.3
16	LLaVA 34B std	120	95.8	15.8	85.8	47.2	66.5	62.2

Reliability varies across families. Qwen models exhibited zero parsing failures and zero truncation in standard mode. LLaMA3 8B in reasoning mode had an NA rate of 10.9%, primarily caused by degenerate repetitive outputs rather than truncation (see Section 3.3).

Task difficulty. Motor vehicle counting showed the narrowest performance spread (5.1 points across all configurations), while frontage length estimation showed the widest (49.7 points), reflecting the inherent difficulty of metric distance estimation from single images and the sensitivity of this task to both model capability and inference mode. Vegetation classification was the hardest task overall, with the highest score being 89.8% proximity and a linearly weighted κ of 0.54 (moderate agreement).

Computation cost. In standard mode, Qwen 7B on a T4 processed each image in 2.17 s, while LLaVA 34B on an A100 required 5.36 s. The gap widened dramatically in reasoning mode: Qwen 7B remained efficient at 16.23 s/image, whereas LLaVA 34B required 160 s/image—nearly 5.3 h for 120 images on an A100. This prohibitive computation time is why LLaVA 34B reasoning was limited to 25 randomly sampled images. These timings highlight that smaller models on consumer-grade GPUs (T4, L4) are attractive for large-scale deployment.

3.3. Practical Limitations

Three practical issues arose during the evaluation. First, Qwen2.5-VL-3B running on a T4 GPU experienced intermittent CUDA device-side assertion failures. The T4 supports only FP16, whereas Qwen models are trained in BF16; the narrower dynamic range of FP16 may lead to numerical overflow during generation, producing invalid tensor indices that crash the GPU context irreversibly. Two images were lost in reasoning mode, and half the standard-mode run was also affected (60 out of 120 images). Running Qwen2.5-VL-3B on a GPU with native BF16 support (e.g., A100 or L4) may resolve this issue.

Second, LLaVA 34B required approximately 160 s per image in reasoning mode on an A100 (5 tasks $\times$ 3 consensus runs $\times$ 1024 max tokens). Consequently, the full 120-image evaluation was performed only in standard mode. In reasoning mode, only a random sample of 25 images was processed to provide a partial estimate of performance. These two partial configurations are marked with crosses in Table 2 and should be interpreted with caution.

Third, the parsing failure rates observed for LLaMA3 8B and Vicuna 7B in reasoning mode reflect architecture-specific limitations in instruction following rather than deficiencies in the prompt design. Analysis of the raw model outputs reveals two distinct failure mechanisms. LLaMA3 8B occasionally enters degenerate repetitive loops, echoing the task definition verbatim without progressing to the expected `ANSWER:` line. Vicuna 7B exhibits a different behaviour: it produces coherent reasoning traces but omits the structured conclusion, ending with a descriptive statement instead of the requested format. Both behaviours occur predominantly in non-truncated responses, confirming that the failures are not caused by insufficient token budget. The same prompts produce near-zero parsing failures with Qwen models and with all LLaVA variants in standard mode.

4. Impact

UVLM addresses a practical gap in the current VLM ecosystem: the absence of a lightweight, accessible tool for systematically comparing Vision–Language Models across architectures using identical evaluation protocols. By abstracting the substantial implementation differences between model families behind a unified interface, UVLM enables reproducible comparisons that would otherwise require significant engineering effort.

The modular package architecture significantly expands UVLM's reach. Local Jupyter support enables researchers working with sensitive data (medical imagery, proprietary datasets, security-related photographs) to run UVLM entirely on-premises without uploading data to cloud services. The programmatic API enables integration into larger automated pipelines—for instance, a remote sensing workflow that downloads satellite imagery, runs UVLM for land-use classification across multiple VLMs, and feeds the consensus results into a GIS database. The modular package structure also makes UVLM suitable as a teaching tool: individual modules (e.g., `parsers.py`, `consensus.py`) can be studied and tested in isolation.

While UVLM was developed in the context of urban research (specifically within the SAGAI [7] and emc2 [8] projects for streetscape analysis) its design is domain-agnostic. The four supported task types (numeric, category, boolean, text) and the free-form prompt builder are potentially applicable to a wide range of image analysis domains, such as medical image interpretation, remote sensing, environmental monitoring, agricultural assessment, infrastructure inspection, and document analysis, though these applications remain to be validated. In each of these potential use cases, the ability to compare multiple VLM architectures on the same task with the same prompts could provide valuable evidence for model selection decisions.

The illustrative example (Section 3) demonstrates that UVLM successfully reveals non-obvious performance patterns (such as smaller models scoring higher than larger ones under identical prompts, and reasoning mode degrading rather than improving certain model families) that would remain hidden without systematic cross-architecture comparison.

5. Conclusions

This paper presented UVLM, a pip-installable Python package that provides a unified interface for loading, configuring, and running Vision–Language Models across architecturally heterogeneous families. The current implementation supports LLaVA-NeXT and Qwen2.5-VL, abstracting their fundamental differences in vision encoding, tokenization, and decoding behind a single inference function. The modular architecture (eight Python modules with no global state), three deployment modes, and backend-agnostic parsing and batch logic make it a practical tool for applied researchers who need to compare VLMs without writing model-specific code.

The illustrative example on 120 street-level images across 16 model configurations demonstrated the software's capabilities: UVLM can structure a complete cross-architecture evaluation and reveal non-obvious patterns (such as a 7B model scoring higher than a 34B model, or reasoning mode degrading performance for certain families) that would remain hidden without a unified evaluation framework. UVLM's truncation detection and consensus validation features further proved effective at flagging reliability issues automatically. These observations are specific to the dataset and prompt configuration used in this study; generalisation would require a larger sample, multiple annotators, and per-model prompt optimisation.

Several directions for future work are envisioned. First, support for additional VLM families—including InternVL [4], BLIP-2 [3], CogVLM, DeepSeek-VL, Molmo, and GLM-V—will be integrated to broaden the comparative scope. Second, multi-GPU batching across images will improve throughput for large-scale evaluations. Third, video frame analysis support will extend UVLM's applicability to temporal visual tasks. On the application side, UVLM will be integrated into the SAGAI workflow [7] as its vision–language inference engine, enabling SAGAI users to compare VLMs and select the most suitable one for their specific streetscape analysis tasks.

Author Contributions: Conceptualization, J.P. and G.F.; methodology, J.P.; software, J.P.; validation, J.P. and G.F.; formal analysis, J.P.; investigation, J.P.; resources, J.P.; data curation, G.F.; writing—original draft preparation, J.P.; writing—review and editing, J.P. and G.F.; visualization, J.P.; supervision, J.P. and G.F.; project administration, G.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the emc2 project under the Driving Urban Transition Partnership, co-funded by ANR (France, grant ANR-23-DUTP-0003), FFG (Austria, grant FO999905461), MUR (Italy, grant 2024/0017648), and Vinnova (Sweden, grant 2023-02581) with contributions from the European Commission.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The UVLM source code (v3.0.2) is publicly available under the Apache 2.0 license at https://github.com/perezjoan/UVLM (accessed on 25 June 2026) and permanently archived on Zenodo (DOI: https://zenodo.org/records/20917845). The repository README includes a minimal reproducible example. The benchmark image dataset is available on Zenodo [15].

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

VLM	Vision–Language Model
UVLM	Unified Vision–Language Model
CoT	Chain-of-Thought
MAE	Mean Absolute Error
GPU	Graphics Processing Unit
CSV	Comma-Separated Values
API	Application Programming Interface

Appendix A. UVLM Benchmark Prompts

Each task is sent as a single prompt: ROLE + TASK + THEORY + FORMAT concatenated. The ROLE is shared across all tasks. Important: When advanced reasoning

is enabled, the FORMAT field is automatically replaced by the code with the chain-of-thought template.

Shared ROLE

You analyze a street-level image. Be precise and follow instructions exactly. The street frontage is the street-facing boundary between the public street space and the adjoining plots, on one side of the street only. It may consist of material elements (building facade, fence, wall, hedge) or immaterial elements (edge of a garden, parking lot, vacant plot, or setback area when no physical boundary exists).

Task 1—Motor Vehicle Counting

Type/Column	task_type: numeric | CSV column: vehicles
ROLE	(shared ROLE above)
TASK	Count the motor vehicles in the image.
THEORY	All motor vehicles must be counted: cars, vans, trucks, buses, streetcars, motorcycles, etc. Do not count bicycles or scooters, even if they have an electric motor. Count all motor vehicles that are visually distinguishable. Motor vehicles partially hidden by other objects should still be considered if clear visual cues are present.
FORMAT (std)	Answer with only one integer number, nothing else.
FORMAT (reas)	(auto-generated by code—do not fill in)

Task 2—Sidewalk Detection

Type/Column	task_type: boolean | CSV column: sidewalk
ROLE	(shared ROLE above)
TASK	Detect the presence of a sidewalk delimiting the street frontage visible in the image.
THEORY	The street frontage extends to the first clearly visible perpendicular intersection. If none is visible, stop at the last visible point of the street-facing boundary. Consider as sidewalk both a raised pavement and a protected pedestrian path bordering the street. If the sidewalk does not cover the entire street frontage, still consider it as present.
FORMAT (std)	Answer with only one word: yes or no
FORMAT (reas)	(auto-generated by code—do not fill in)

Task 3—Pedestrian Entrance Counting

Type/Column	task_type: numeric | CSV column: entrances
ROLE	(shared ROLE above)
TASK	Count all pedestrian entrances to the plots abutting the street frontage visible in the image.

THEORY	The street frontage extends to the first clearly visible perpendicular intersection. If none is visible, stop at the last visible point of the street-facing boundary. A pedestrian entrance is either: - A door in a building (including shop entrances) giving direct access from the street, or - A pedestrian gate in a fence or wall giving access from the street to an enclosed open area. Garage doors and car-only gates are NOT pedestrian entrances. Exception: if a car gate is the only access to a fenced plot and has a doorbell, street number, or mailbox, count it as one pedestrian entrance. If a pedestrian gate gives access to an enclosed open space before a building, count only the gate, not the building doors behind it. Pedestrian entrances partially hidden by foreground objects (trees, cars, people, street furniture) should still be considered if clear visual cues are present.
FORMAT (std)	Answer with only one integer number, nothing else.
FORMAT (reas)	(auto-generated by code—do not fill in)

Task 4—Street Frontage Length

Type/Column	task_type: numeric | CSV column: length
ROLE	(shared ROLE above)
TASK	Estimate the total length (in meters) of the street frontage visible in the image.
THEORY	The street frontage extends to the first clearly visible perpendicular intersection. If none is visible, stop at the last visible point of the street-facing boundary. The length is the sum of the linear lengths of all material and immaterial boundary elements along the street, from the left edge to the right edge of the visible street frontage. Only the street-facing boundary line is measured. Elements behind this boundary (such as building facades behind setbacks) are not included in the length, but may be used as visual references. Choose ONE type of standard-size reference object visible near the street-facing boundary and use it consistently to estimate the full length: - Cars (4 to 4.5 m each) - Parking bays (about 5 m each) - Doors or windows (about 1 m wide each) - Sidewalk slabs or modules - Building stories projected horizontally (about 3 m per story) - Electricity posts (about 8 m tall) or streetlamps (4 to 8 m tall), projected onto the horizontal boundary line. Account for perspective: objects farther from the camera appear shorter than their actual size.
FORMAT (std)	Answer with only one integer number (meters), nothing else.
FORMAT (reas)	(auto-generated by code—do not fill in)

Task 5—Vegetation Classification

Type/Column	task_type: numeric | CSV column: vegetation_type
ROLE	(shared ROLE above)
TASK	Classify the vegetation presence along the street frontage visible in the image into one of six types.

THEORY	The street frontage extends to the first clearly visible perpendicular intersection. If none is visible, stop at the last visible point of the street-facing boundary. First determine two things: A. Trees in the street space: Trees are in the street space only if located in the street right-of-way along the frontage (on the sidewalk, curb strip, planting strip, or median—between the carriageway and the plot boundary). Trees inside plots (behind the plot boundary) do NOT count as trees in the street space. Only count trees large enough to form a canopy that at least partially covers the street space. If the image was taken in winter, deciduous trees may have no leaves. Still count them as trees if their trunk and branch structure indicate they are large enough to produce a canopy during the growing season. B. Vegetation in plots: Vegetation in plots refers to trees, shrubs, lawn, or hedges seen behind the street-facing boundary inside the abutting plots. In winter, consider deciduous trees and shrubs as vegetation even if leafless. Then select the matching type: Type 1 = Trees in street space YES + Plots highly vegetated (almost entirely covered) Type 2 = Trees in street space NO + Plots highly vegetated Type 3 = Trees in street space YES + Plots partially vegetated (vegetated setbacks or side gardens) Type 4 = Trees in street space NO + Plots partially vegetated Type 5 = Trees in street space YES + Plots little or no vegetation Type 6 = Trees in street space NO + Plots little or no vegetation.
FORMAT (std)	Answer with only one integer number (1 to 6), nothing else.
FORMAT (reas)	(auto-generated by code—do not fill in)

Appendix B. Benchmark Results and Methodology

Dataset

The benchmark dataset contains 120 street-level images of French urban frontages (IDs D01–D48, N01–N04, S01–S68) [15]. Ground truth annotations were produced by a single expert annotator (Fusco, G.). Some images were excluded from Qwen-3B runs due to CUDA crashes on a T4 GPU. Due to computational cost constraints on an A100 GPU, only 25 images were randomly evaluated for LLaVA-34B reasoning mode.

Human annotations

Each entry below follows the format `ID[T1,T2,T3,T4,T5]` where T1 = motor vehicles, T2 = sidewalk (1/0), T3 = pedestrian entrances, T4 = frontage length (m), T5 = vegetation type (1–6).

D01[0,1,2,12,6], D02[4,1,2,20,4], D03[2,1,0,20,3], D04[3,1,5,20,6], D05[0,1,3,12,6], D06[3,1,6,28,6], D07[0,0,0,12,4], D08[0,1,3,12,6], D09[2,0,1,24,6], D10[0,1,1,12,4], D11[4,1,6,28,6], D12[3,1,5,30,6], D13[7,1,3,40,4], D14[1,1,1,30,4], D15[5,1,5,30,6], D16[2,1,2,23,4], D17[1,1,3,12,6], D18[5,1,0,20,5], D19[0,1,1,8,4], D20[0,1,1,10,4], D21[4,0,2,10,4], D22[0,1,2,13,4], D23[0,0,1,16,4], D24[0,0,1,10,4], D25[0,0,0,15,4], D26[0,0,1,20,4], D27[0,0,1,18,4], D28[0,0,1,22,4], D29[4,0,2,30,6], D30[0,0,3,14,6], D31[7,1,4,40,4], D32[2,0,2,18,5], D33[3,1,2,18,6], D34[4,1,2,18,6], D35[1,1,2,20,4], D36[0,0,2,20,4], D37[0,0,1,18,4], D38[1,1,2,25,4], D39[3,1,4,15,6], D40[2,1,2,21,6], D41[0,1,1,9,4], D42[0,1,6,19,5], D43[0,1,4,18,5], D44[5,1,1,25,4], D45[1,1,2,10,6], D46[0,0,1,9,4], D47[0,0,2,25,4], D48[0,0,1,25,4], N01[3,1,0,10,1], N02[2,1,0,11,2], N03[3,1,0,12,3], N04[3,1,0,8,1], S01[3,1,2,12,6], S02[1,1,4,25,6], S03[0,1,2,8,6], S04[1,1,4,25,6], S05[0,1,3,16,6], S06[2,1,5,25,6], S07[11,1,3,18,5], S08[0,1,2,9,6], S09[1,1,2,13,6], S10[0,1,4,10,6], S11[2,1,1,9,6], S12[2,1,3,16,6], S13[2,1,2,15,6], S14[2,1,3,11,6], S15[5,1,2,18,4], S16[2,1,2,15,6], S17[1,1,1,7,6], S18[2,1,1,8,6], S19[3,1,2,12,6], S20[3,1,2,20,6], S21[3,1,1,15,6], S22[4,1,3,18,6], S23[2,1,1,11,5], S24[6,1,2,15,6], S25[3,1,1,8,6], S26[0,1,2,15,6], S27[3,1,2,15,4], S28[3,1,1,18,4], S29[1,1,2,15,4], S30[1,1,1,14,4], S31[0,1,3,13,4], S32[1,1,2,15,4], S33[1,1,3,25,4], S34[2,1,6,40,4], S35[1,1,1,15,4], S36[5,1,4,30,4], S37[0,1,0,12,6], S38[0,1,0,15,3], S39[1,1,0,10,4], S40[1,0,1,15,4], S41[3,1,0,18,3], S42[1,1,3,18,4], S43[1,1,2,30,4], S44[0,1,1,9,4],

S45[0,1,1,15,4], S46[2,1,0,12,4], S47[1,1,0,26,4], S48[3,1,4,12,6], S49[4,1,2,14,6], S50[1,1,0,20,4], S51[1,1,3,24,3], S52[4,1,6,25,6], S53[0,1,3,40,5], S54[0,0,1,10,2], S55[2,1,0,20,3], S56[0,1,0,12,2], S57[3,1,8,32,5], S58[0,1,0,12,1], S59[1,1,0,30,2], S60[1,1,1,24,2], S61[0,1,5,28,3], S62[0,1,0,24,4], S63[1,1,1,13,2], S64[2,1,1,20,3], S65[0,1,1,16,5], S66[0,1,0,18,3], S67[1,1,0,15,1], S68[0,1,0,20,2].

Tasks

The benchmark evaluates the five visual interpretation tasks detailed in Appendix A.

Inference Modes

Two inference configurations are evaluated. Standard mode: direct answer generation. Maximum generation length: 50 tokens. Each task is executed three times, and the final prediction is obtained using majority vote (consensus). Reasoning mode: chain-of-thought reasoning followed by a final answer. Maximum generation length: 1024 tokens. The model is instructed to reason step-by-step and end with: ANSWER: <value>. Three independent runs are executed and aggregated using majority vote.

Evaluation Metrics

Binary classification (Sidewalk): Accuracy: Fraction of images where the model prediction matches the human annotation. Sensitivity: True positive rate: proportion of sidewalk-present images correctly detected. Specificity: True negative rate: proportion of sidewalk-absent images correctly identified. Cohen's κ: Chance-corrected agreement between model and human annotations.

Continuous and counting tasks: MAE (Mean Absolute Error): Average magnitude of prediction errors. Lower values indicate better performance. Bias: Mean signed error (model $-$ human). Positive values indicate systematic overestimation; negative values indicate underestimation. MAPE (Mean Absolute Percentage Error): MAE expressed as a percentage of the true value. Particularly relevant for frontage length where absolute errors scale with size. Exact match: Fraction of images where the model prediction exactly matches the human value. $\pm 1/\pm 2$ tolerance: Fraction of predictions within ± 1 or ± 2 units of the human value. ± 10 m tolerance: Equivalent tolerance metric for frontage length. Pearson correlation (r): Measures the linear association between model and human values.

Ordinal classification (Vegetation): Weighted Cohen's κ (linear weights): Measures agreement for ordered categories while penalizing large misclassifications more strongly than small ones. Interpretation follows the same scale as standard κ.

Proximity Score: The main model comparison indicator is the Proximity Score. The range corresponds to the observed human value range for each task (e.g., 0–8 vehicles, 4–60 m frontage, 1–6 vegetation classes). Unlike Exact Match, this metric rewards near-correct predictions. The Overall Proximity Score is computed as the unweighted mean across the five tasks and is used as the primary model comparison criterion.

Reasoning Impact: To evaluate the effect of chain-of-thought reasoning, we compute: NA count: Number of consensus runs where the parser fails to extract a valid answer. Truncation count: Number of task $\times$ image pairs where generation reached the maximum token limit. Truncation does not necessarily imply an incorrect answer, but it increases the probability of parsing failures because the reasoning chain may be incomplete.

Table A1. Cost-performance analysis (* computing costs (CC) based on Google Colab GPU). Green indicates best values; red indicates worst values. The same colour convention applies to Tables A2–A7.

Model	GPU	Load (min)	Mode	Sec/Image	CC */Image	Proximity
Qwen 3B	T4	0.61	Standard	0.73	0.00033	76.0%
Qwen 3B	T4	0.61	Reasoning	17.23	0.00644	76.1%
Qwen 7B	T4	1.11	Standard	2.17	0.00090	75.4%
Qwen 7B	T4	1.11	Reasoning	16.23	0.00779	77.4%
Qwen 32B	A100	3.66	Standard	1.06	0.00519	83.3%
Qwen 32B	A100	3.66	Reasoning	60.50	0.14031	88.0%
LLaVA Mistral 7B	T4	1.11	Standard	5.60	0.00269	74.4%
LLaVA Mistral 7B	T4	1.11	Reasoning	26.31	0.01401	73.3%
LLaVA Vicuna 7B	T4	1.07	Standard	3.03	0.00314	83.1%
LLaVA Vicuna 7B	T4	1.07	Reasoning	17.99	0.04031	77.0%
LLaMA3 LLaVA 8B	L4	1.20	Standard	1.62	0.00029	65.3%
LLaMA3 LLaVA 8B	L4	1.20	Reasoning	27.16	0.01522	71.4%
LLaVA Vicuna 13B	A100	1.63	Standard	1.19	0.00314	77.9%
LLaVA Vicuna 13B	A100	1.63	Reasoning	19.30	0.04031	77.0%
LLaVA 34B	A100	3.70	Standard	5.36	0.01191	62.2%
LLaVA 34B	A100	3.70	Reasoning	160.01	0.56122	74.2%

Table A2. Task 1: Motor Vehicles.

Model	N	MAE	Bias	Exact	±1	Correlation	Proximity
Qwen 3B std	60	0.38	−0.38	81.7%	91.7%	0.90	96.5%
Qwen 3B reas	118	0.43	−0.28	70.3%	92.4%	0.88	96.1%
Qwen 7B std	120	0.28	−0.21	80.8%	95.8%	0.92	97.5%
Qwen 7B reas	120	0.47	−0.43	70.8%	92.5%	0.87	95.8%
Qwen 32B std	120	0.21	−0.11	81.7%	98.3%	0.96	98.1%
Qwen 32B reas	120	0.25	−0.20	81.7%	95.0%	0.95	97.7%
LLaVA Mis7B std	120	0.47	−0.40	68.3%	90.8%	0.88	95.8%
LLaVA Mis7B reas	119	0.66	−0.51	57.1%	85.7%	0.82	94.0%
LLaVA Vic7B std	120	0.51	−0.14	63.3%	92.5%	0.87	95.4%
LLaVA Vic7B reas	118	0.77	−0.19	54.2%	83.1%	0.78	93.0%
LLaMA3 8B std	120	0.45	−0.32	67.5%	92.5%	0.89	95.9%
LLaMA3 8B reas	120	0.69	−0.09	55.8%	86.7%	0.76	93.7%
LLaVA Vic13B std	120	0.43	−0.23	73.3%	92.5%	0.85	96.1%
LLaVA Vic13B reas	120	0.71	−0.44	58.3%	83.3%	0.77	93.6%
LLaVA 34B std	120	0.47	−0.28	73.3%	91.7%	0.74	95.8%
LLaVA 34B reas	25	1.80	1.56	64.0%	88.0%	0.72	88.0%

Table A3. Task 2: Sidewalk.

Model	N	Accuracy	Sensitivity	Specificity	Cohen κ	Proximity
Qwen 3B std	60	75.0%	93.0%	29.4%	0.27	75.0%
Qwen 3B reas	118	56.8%	63.6%	21.1%	−0.11	56.8%
Qwen 7B std	120	40.0%	31.7%	84.2%	0.07	40.0%
Qwen 7B reas	120	62.5%	62.4%	63.2%	0.15	62.5%
Qwen 32B std	120	80.8%	90.1%	31.6%	0.23	80.8%
Qwen 32B reas	120	85.0%	97.0%	21.1%	0.24	85.0%
LLaVA Mis7B std	120	45.0%	42.6%	57.9%	0.00	45.0%
LLaVA Mis7B reas	120	70.0%	74.3%	47.4%	0.16	70.0%
LLaVA Vic7B std	120	84.2%	100.0%	0.0%	0.00	84.2%
LLaVA Vic7B reas	120	77.5%	85.1%	36.8%	0.21	77.5%
LLaMA3 8B std	120	51.7%	43.6%	94.7%	0.17	51.7%
LLaMA3 8B reas	120	46.7%	38.6%	89.5%	0.12	46.7%
LLaVA Vic13B std	120	80.8%	93.1%	15.8%	0.11	80.8%
LLaVA Vic13B reas	120	65.8%	68.3%	52.6%	0.14	65.8%
LLaVA 34B std	120	15.8%	0.0%	100.0%	0.00	15.8%
LLaVA 34B reas	25	80.0%	95.0%	20.0%	0.19	80.0%

Table A4. Task 3: Pedestrian Entrances.

Model	N	MAE	Bias	Exact	±1	Correlation	Proximity
Qwen 3B std	60	1.35	−1.28	26.7%	61.7%	0.67	77.5%
Qwen 3B reas	118	1.16	−0.92	37.3%	72.9%	0.44	85.5%
Qwen 7B std	120	0.88	−0.67	45.0%	79.2%	0.71	89.0%
Qwen 7B reas	120	1.24	−0.59	34.2%	70.8%	0.56	84.9%
Qwen 32B std	120	0.62	−0.28	50.0%	90.8%	0.84	92.3%
Qwen 32B reas	120	0.77	−0.42	49.2%	84.2%	0.71	90.4%
LLaVA Mis7B std	120	1.14	−0.91	33.3%	74.2%	0.55	85.7%
LLaVA Mis7B reas	120	1.30	−0.52	32.5%	69.2%	0.16	83.9%
LLaVA Vic7B std	120	1.14	−0.29	31.7%	75.8%	0.27	85.7%
LLaVA Vic7B reas	120	1.53	−0.44	22.5%	52.5%	0.32	80.9%
LLaMA3 8B std	120	1.12	0.43	26.7%	71.7%	0.59	86.0%
LLaMA3 8B reas	117	1.21	0.38	29.1%	71.8%	0.55	84.9%
LLaVA Vic13B std	120	1.10	−0.08	26.7%	77.5%	0.47	86.3%
LLaVA Vic13B reas	120	1.54	0.26	25.0%	57.5%	0.28	80.7%
LLaVA 34B std	120	1.13	0.62	23.3%	75.0%	0.60	85.8%
LLaVA 34B reas	25	2.64	1.60	36.0%	56.0%	0.15	76.5%

Table A5. Task 4: Street Frontage Length.

Model	N	MAE (m)	Bias (m)	MAPE %	±10 m	Correlation	Proximity
Qwen 3B std	60	11.4	6.1	81.8	73.3%	−0.14	70.1%
Qwen 3B reas	118	13.0	5.2	81.5	61.9%	0.08	66.5%
Qwen 7B std	120	10.5	7.7	63.8	74.2%	0.41	70.1%
Qwen 7B reas	120	9.0	−6.2	47.3	69.2%	0.29	73.3%
Qwen 32B std	120	16.5	16.0	106.1	50.0%	0.49	55.4%
Qwen 32B reas	120	5.8	0.1	35.8	85.0%	0.48	82.5%
LLaVA Mis7B std	120	10.9	0.9	53.7	74.2%	0.30	75.4%
LLaVA Mis7B reas	120	12.5	−5.5	65.9	52.5%	0.13	65.9%
LLaVA Vic7B std	120	9.5	−2.1	48.1	76.7%	0.06	76.5%
LLaVA Vic7B reas	120	53.2	37.8	333.8	55.0%	−0.02	67.1%
LLaMA3 8B std	120	42.9	41.8	258.4	24.2%	0.41	32.8%
LLaMA3 8B reas	120	11.7	−6.6	66.7	53.3%	0.07	66.3%
LLaVA Vic13B std	116	27.7	21.3	150.1	56.0%	0.36	56.0%
LLaVA Vic13B reas	120	10.8	−5.1	58.7	65.0%	0.13	70.4%
LLaVA 34B std	120	17.7	−17.7	98.3	16.7%	−0.03	47.2%
LLaVA 34B reas	25	10.6	2.9	65.8	72.0%	0.22	70.4%

Table A6. Task 5: Vegetation Type.

Model	N	Exact	±1 Class	MAE Class	κ Linear	Proximity
Qwen 3B std	60	38.3%	43.3%	1.97	0.26	60.7%
Qwen 3B reas	118	37.3%	57.6%	1.23	0.31	75.4%
Qwen 7B std	120	47.5%	63.3%	0.97	0.42	80.7%
Qwen 7B reas	120	26.7%	36.7%	1.47	0.18	70.7%
Qwen 32B std	120	68.3%	80.8%	0.51	0.64	89.8%
Qwen 32B reas	120	55.8%	66.7%	0.79	0.54	84.2%
LLaVA Mis7B std	120	27.5%	55.0%	1.48	0.25	70.3%
LLaVA Mis7B reas	120	13.3%	29.2%	2.36	0.00	52.8%
LLaVA Vic7B std	120	15.8%	73.3%	1.30	−0.02	74.0%
LLaVA Vic7B reas	120	27.5%	47.5%	1.68	0.10	66.3%
LLaMA3 8B std	120	16.7%	42.5%	1.98	0.00	60.3%
LLaMA3 8B reas	120	15.0%	48.3%	1.77	0.19	65.3%
LLaVA Vic13B std	120	20.0%	60.8%	1.48	0.05	70.3%
LLaVA Vic13B reas	120	30.8%	60.8%	1.28	0.21	74.5%
LLaVA 34B std	120	10.8%	54.2%	1.68	0.05	66.5%
LLaVA 34B reas	25	16.0%	40.0%	2.36	0.06	56.0%

Table A7. Reasoning vs Standard: Proximity delta per task.

Model Pair	Vehicles	Sidewalk	Entrances	Length	Vegetation	Mean Δ	Verdict
Qwen 3B std → Qwen 3B reas	−0.4%	−18.2%	+8.0%	−3.6%	+14.8%	+0.1%	Neutral
Qwen 7B std → Qwen 7B reas	−1.7%	+22.5%	−4.1%	+3.2%	−10.0%	+2.0%	Helps
Qwen 32B std → Qwen 32B reas	−0.4%	+4.2%	−1.9%	+27.1%	−5.7%	+4.7%	Helps
LLaVA Mis7B std → LLaVA Mis7B reas	−1.8%	+25.0%	−1.9%	−9.5%	−17.5%	−1.1%	Hurts
LLaVA Vic7B std → LLaVA Vic7B reas	−2.4%	−6.7%	−4.8%	−9.4%	−7.7%	−6.2%	Hurts
LLaMA3 8B std → LLaMA3 8B reas	−2.2%	−5.0%	−1.1%	+33.5%	+5.0%	+6.0%	Helps
LLaVA Vic13B std → LLaVA Vic13B reas	−2.6%	−15.0%	−5.5%	+14.4%	+4.2%	−0.9%	Neutral
LLaVA 34B std → LLaVA 34B reas	−7.8%	+64.2%	−9.3%	+23.1%	−10.5%	+11.9%	Helps

Table A8. Parsing failures (NA counts across all consensus runs).

Model	N Images	Vehicles	Sidewalk	Length	Entrances	Vegetation	Total NAs	NA Rate
Qwen 3B std	60	0	0	0	0	0	0	0.0%
Qwen 3B reas	118	11	0	1	2	0	14	0.8%
Qwen 7B std	120	0	0	0	0	0	0	0.0%
Qwen 7B reas	120	0	0	0	0	0	0	0.0%
Qwen 32B std	120	0	0	0	0	0	0	0.0%
Qwen 32B reas	120	0	0	0	0	0	0	0.0%
LLaVA Mis7B std	120	0	0	0	0	0	0	0.0%
LLaVA Mis7B reas	120	9	0	8	26	0	43	2.4%
LLaVA Vic7B std	120	0	0	0	0	0	0	0.0%
LLaVA Vic7B reas	120	87	0	9	13	1	110	6.1%
LLaMA3 8B std	120	0	0	5	0	0	5	0.3%
LLaMA3 8B reas	120	68	0	0	87	41	196	10.9%
LLaVA Vic13B std	120	0	0	80	0	0	80	4.4%
LLaVA Vic13B reas	120	18	0	11	13	0	42	2.3%
LLaVA 34B std	120	0	0	0	1	0	1	0.1%
LLaVA 34B reas	25	0	0	3	1	1	5	1.3%

Table A9. Truncation alerts.

Model	N Images	Vehicles	Sidewalk	Length	Entrances	Vegetation	Total	Trunc. Rate
Qwen 3B std	60	0	0	0	0	0	0	0.0%
Qwen 3B reas	118	1	0	0	1	0	2	0.3%
Qwen 7B std	120	0	0	0	0	0	0	0.0%
Qwen 7B reas	120	0	0	3	0	0	3	0.5%
Qwen 32B std	120	0	0	0	0	0	0	0.0%
Qwen 32B reas	120	0	0	0	0	0	0	0.0%
LLaVA Mis7B std	120	0	0	0	0	0	0	0.0%
LLaVA Mis7B reas	120	0	0	10	9	0	19	3.2%
LLaVA Vic7B std	120	0	0	0	0	0	0	0.0%
LLaVA Vic7B reas	120	0	0	4	1	0	5	0.8%
LLaMA3 8B std	120	0	0	3	0	0	3	0.5%
LLaMA3 8B reas	120	9	30	11	31	20	101	16.8%
LLaVA Vic13B std	120	0	0	21	0	0	21	3.5%
LLaVA Vic13B reas	120	0	0	6	1	1	8	1.3%
LLaVA 34B std	120	0	0	3	1	0	4	0.7%
LLaVA 34B reas	25	0	1	3	2	0	6	4.8%

References

1. Liu, H.; Li, C.; Wu, Q.; Lee, Y.J. Improved Baselines with Visual Instruction Tuning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 17–21 June 2024; pp. 26296–26306.
2. Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. Qwen2.5-VL Technical Report. *arXiv* **2025**, arXiv:2502.13923.
3. Li, J.; Li, D.; Savarese, S.; Hoi, S. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In Proceedings of the 40th International Conference on Machine Learning (ICML 2023), Honolulu, HI, USA, 23–29 July 2023.
4. Chen, Z.; Wu, J.; Wang, W.; Su, W.; Chen, G.; Xing, S.; Zhong, M.; Zhang, Q.; Zhu, X.; Lu, L.; et al. InternVL: Scaling up Vision Foundation Models and Aligning for Generic Visual-Linguistic Tasks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024), Seattle, WA, USA, 17–21 June 2024.
5. Li, B.; Zhang, Y.; Guo, D.; Zhang, R.; Li, F.; Zhang, H.; Zhang, K.; Zhang, P.; Li, Y.; Liu, Z.; et al. LLaVA-OneVision: Easy Visual Task Transfer. *arXiv* **2024**, arXiv:2408.03326.
6. Radford, A.; Kim, J.W.; Hallacy, C.; Ramesh, A.; Goh, G.; Sastry, S.A.; Askell, A.; Mishkin, P.; Clark, J.; Krueger, G.; et al. Learning Transferable Visual Models From Natural Language Supervision. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), Virtual, 18–24 July 2021; pp. 8748–8763.
7. Perez, J.; Fusco, G. Streetscape Analysis with Generative AI (SAGAI): Vision-language assessment and mapping of urban scenes. *Geomatica* **2025**, *77*, 100063.
8. Fusco, G.; Berghauser Pont, M.; Cutini, V.; Psenner, A. Guiding principles for the 15-min city in peripheral areas: The emc2 model. In Proceedings of AESOP 2024, Paris, France, 8–12 July 2024; pp. 690–707.
9. Perez, J.; Fusco, G. Population potential on catchment area (PPCA): A Python-based tool for worldwide geospatial population analysis. *SoftwareX* **2025**, *31*, 102245.
10. Duan, H.; Yang, J.; Qiao, Y.; Fang, X.; Chen, L.; Liu, Y.; Dong, X.; Zang, Y.; Zhang, P.; Wang, J.; et al. VLMEvalKit: An Open-Source Toolkit for Evaluating Large Multi-Modality Models. In Proceedings of the 32nd ACM International Conference on Multimedia (MM'24), Melbourne, Australia, 28 October–1 November 2024; pp. 11198–11201.
11. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-Art Natural Language Processing. In Proceedings of EMNLP 2020: System Demonstrations, Virtual, 16–20 November 2020; pp. 38–45.
12. Dettmers, T.; Lewis, M.; Belkada, Y.; Zettlemoyer, L. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In Proceedings of theAdvances in Neural Information Processing Systems 35 (NeurIPS 2022), New Orleans, LA, USA, 28 November–9 December 2022.
13. Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Ichter, B.; Xia, F.; Chi, E.H.; Le, Q.V.; Zhou, D. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Proceedings of the Advances in Neural Information Processing Systems 35 (NeurIPS 2022), New Orleans, LA, USA, 28 November–9 December 2022.
14. Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; Narang, S.; Chowdhery, A.; Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In Proceedings of the 11th International Conference on Learning Representations (ICLR 2023), Kigali, Rwanda, 1–5 May 2023.
15. Fusco, G.; Perez, J.; Picard, G.; Spieck, L. Sample of Street View Images of French Urban Areas: Drap, Nice, Seclin. Zenodo. 2026. Available online: https://zenodo.org/records/18959690 (accessed on 25 June 2026).