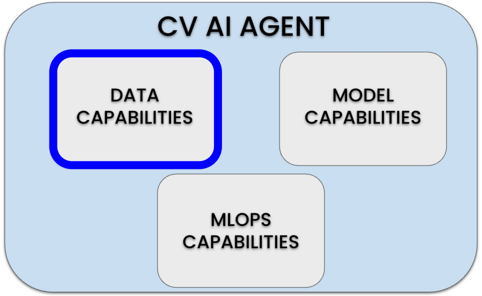
AgenticVision

User Journey of the agentic CV platform

HTX User Requirements/Capabilities (Data-specific only)

This user journey maps out how users interact with the agentic data management system. The system is capable of performing the following :

1. **Data preprocessing :** Reduces the current 70% of time HTX currently spends on data processing, curation, and cleaning
2. **Data Annotation :** Resolves inconsistent data standardization and labeling issues (e.g., swimmer definitions, boat classifications)
3. **Data Quality :** Automates quality assessment of visual assets
4. **Data Augmentation and generation :** Augments vision datasets for data balancing and model generalization capabilities.
5. **Search :** Enables efficient search and retrieval across video repositories



User journey

Broken down into three main aspects :

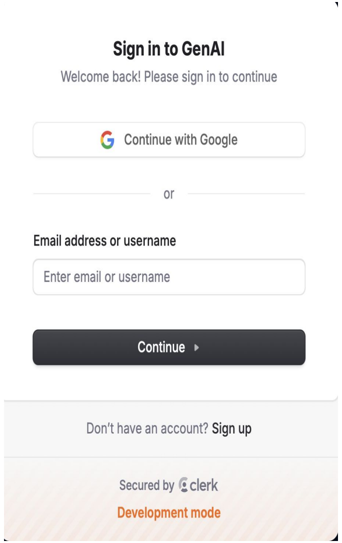
1. Initial setup and configuration
2. Automatic mode detection and tasks generation
3. Execution flows

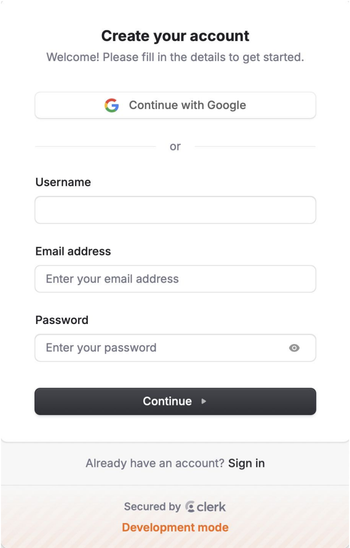
Phase 1 : Initial Setup & Configuration

1. System onboarding - login

User Actions: logging in

- Access a web interface interface hosted on a server, show login page and authenticate using clerk (left image)
- Create profile and complete setup if account not created using clerk (right image)



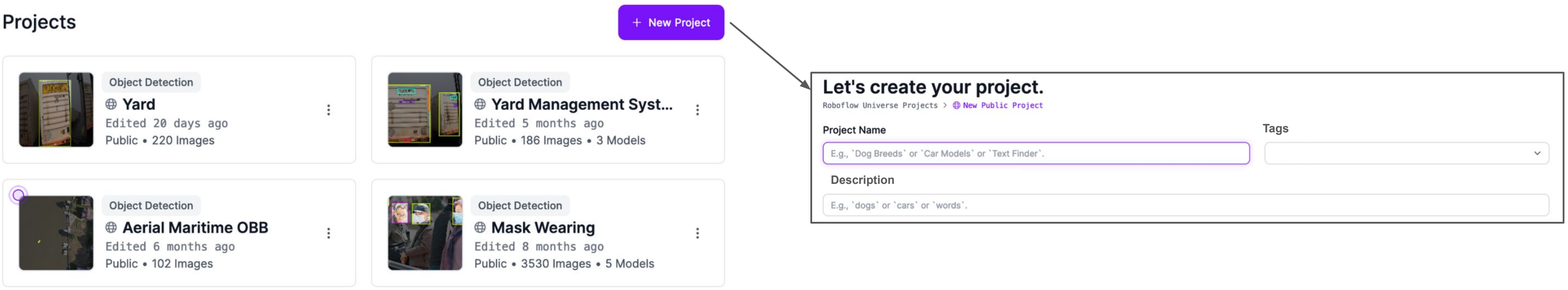


Source : AI Coach CMS

1. System onboarding - Projects

User actions: Creating new project

- Show a list of all projects
- A project contains dataset and the associated models
- Datasets and models will be versioned within each project
- User Clicks on “New Project” to create a project

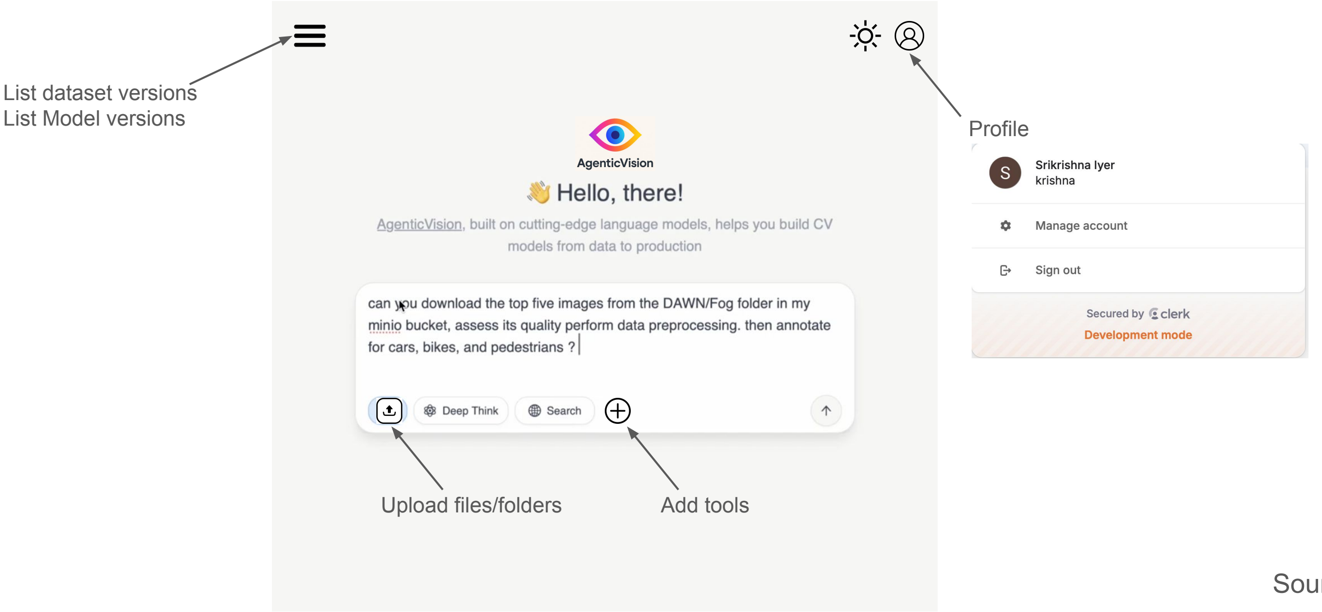


Source : roboflow

1. System onboarding - Chat window

System Response: First page

- Show welcome interface with a textbox.
- Buttons from left to right : upload folder, Deep think (TBD), Search (TBD), tools (show existing tools, modify tools)
- Clicking the “+” icon, takes you to a list of available tools and how you can add/modify/delete them

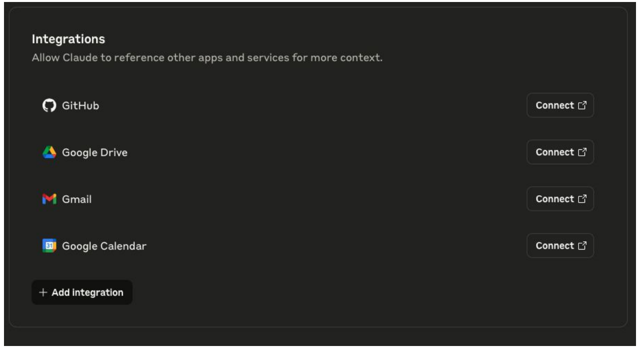


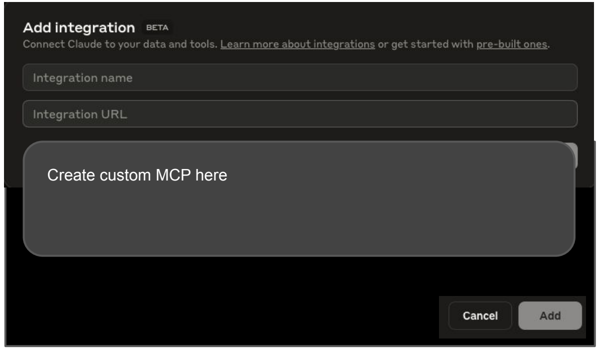
Source : [langmanus demo]()

1. System onboarding - Adding custom tools

User Actions: Adding tools

- Clicking on the “+” icon
- Add and modify MCP tools



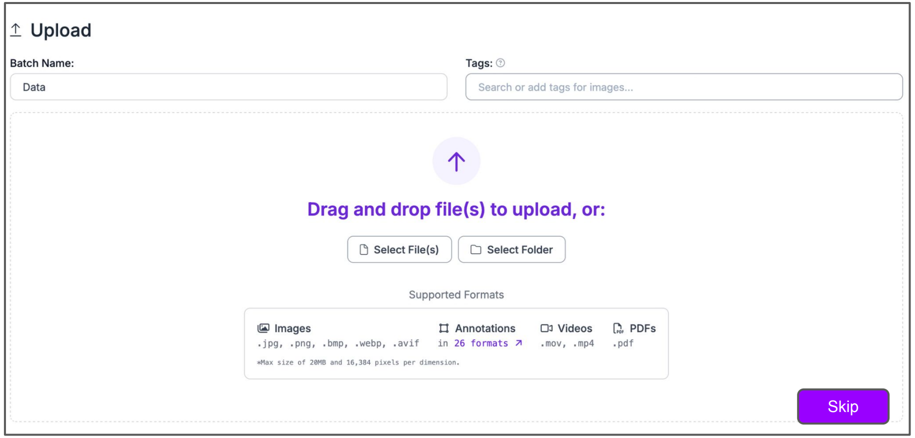


Source : claude UI MCP

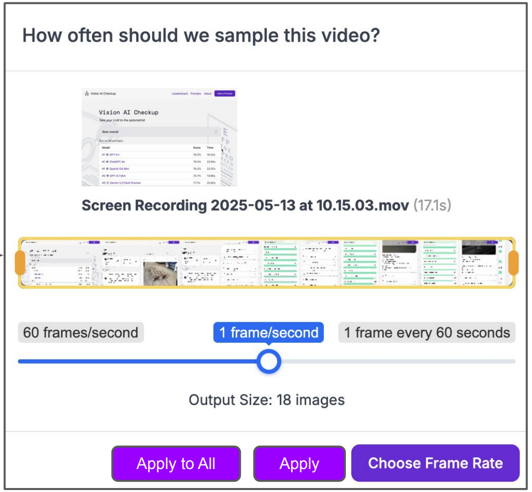
1. System onboarding - Upload files/folders

User Actions: Uploading data

- Clicking on the “upload” icon to open a pop up menu
- If uploading videos, the system should automatically detect .mp4 files and pop up an additional menu to select fps (right image)



If .mp4 detected



Source : [roboflow]

Phase 2 : automatic Mode detection and task generation

2. Intelligent mode detection

System Action: Mode detection

A planner agent (reasoning LLM eg. o3) will determine which type of mode is triggered based on user input :

a. **Fully Autonomous Triggers:**
    - Open-ended natural language requests
    - Minimal technical specifications
    - General goals without specific methodologies
    - Examples: “Collect Singapore traffic datasets”, "Label these images", "Clean up this dataset", "Make these photos better", “Annotate the images”
b. **User Specified Triggers:**
    - Detailed technical requirements
    - Specific tool mentions or parameters
    - Explicit pipeline steps mentioned
    - Examples: "Use YOLO for detection with confidence threshold 0.85", "Apply Gaussian blur with sigma=2.1"
c. **Hybrid Triggers:**
    - Partial specifications with room for optimization
    - Requests for collaboration or recommendations
    - Mixed technical and natural language elements
    - Quality requirements with flexible implementation
    - Examples: "Improve quality while maintaining file sizes under 2MB", “Denoise the images but also check if they can be improved further”, “apply cutmix, resize images to 128x128 but also try other data augmentation techniques”

2. Task generation

Phase 3 : Execution Flows

Backend : Workflow Orchestration

The system should support a “**combination**” of two workflows :

1. **Static Workflows (RPA):** Predefined sequences designed for repetitive tasks.

Tool	Data Ingestion	Data Processing	Data Annotation	Augmentation	Data Quality Assessment	Automated CI/CD	Method
Roboflow	✔️ (image uploads APIs/CLI)	✔️ (auto preproc pipeline)	✔️ (built-in labeling)	✔️ (resize, flip, color augment)	🛠️ (via scripts + validation API)	🛠️ (via API + CI pipelines)	low-code + APIs/CLI
Airflow	✔️ (operators for storage, HTTP, etc.)	✔️ (Python tasks/operators)	❌	🛠️ (scripted within DAGs)	🛠️ (via hooks, custom tasks)	✔️ (native DAG scheduling + triggers)	code (Python)
Kubeflow	✔️ (via Pipelines/KFServing)	✔️ (components, pipelines)	❌ (needs script or plugin)	🛠️ (script in components)	🛠️ (via MLMD, metrics)	✔️ (Pipeline orchestration + CI/CD hooks using kubernetes)	code (Python, YAML)

2. **Agentic Workflows:** Dynamic workflows capable of adapting to different conditions. For some use cases, calling static workflows as an API or python function via MCP.

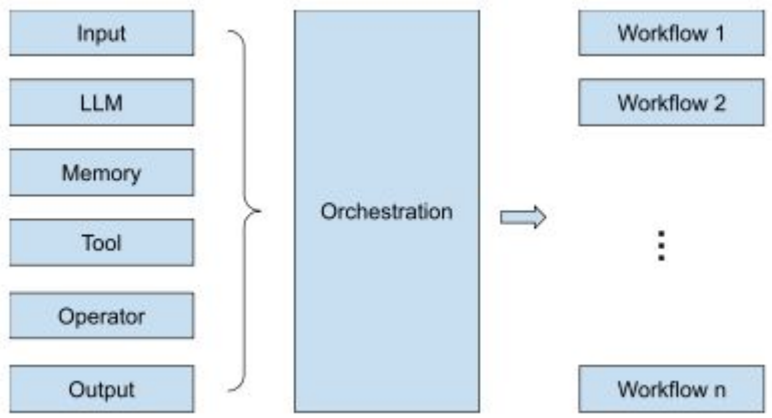
Tool	Long-running Tasks	System Integration	Human-in-the-Loop	Method
LangGraph	✅ Yes	✅ Yes (LangServe)	✅ Yes (supports human-in-loop)	Code (Python, graph-based orchestrator)
Dify	✅ Yes (workflow engine, celery powered)	✅ Yes (connects LLMs & APIs via UI & API)	❌ No	Low-code (UI workflows + prompt IDE)
n8n	✅ Yes (via external async, supports workflows)	✅ Yes (800+ connectors, HTTP node)	✅ via Human-in-the-Loop node (paid plan)	Low-code / No-code + JS/TS scripting
CrewAI	✅ Yes (Flow engine/Celery-powered)	❌ No (no built-in system integrations; but tool/plugin support)	❌Partial (via plugin/UI, but limited interactivity)	Code (Python) + some no-code Studio

Backend : Workflow Orchestration

Components

Each workflow may include the following components:

- **Input:** Data sources or user interactions that initiate the workflow.
- **LLM/VLM:** Language/Vision-language models for understanding and processing tasks.
- **Memory:** Context retention to support long-term reasoning.
- **MCP Tool integration:** External APIs, functions or static workflows that perform specific actions.
- **Operator:** Logic that controls the workflow's flow and decision-making.
- **Output:** Final results or actions produced by the workflow.

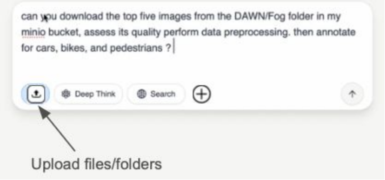


3.1 : **Execution Flows ~** Data collection

System Actions : Data collection

- Mode dependent data collection
  - Fully autonomous
    - Open-ended request to collect datasets
    - Use browser-use agent that uses a headless browser to emulate mouse clicks to download datasets
    - The system combines datasets and their annotations from multiple sources. Eg. Huggingface, Kaggle
  - User specified
    - Upload data manually within the UI
    - Add MCP tool connection to external storage. Eg. MinIO, S3
    - System makes sure annotation formats are aligned to user request.
  - Hybrid
    - Upload data manually within the UI
    - User can also request additional data request and combine with the uploaded data. For eg. “Collect 10K samples of hazy traffic data to augment my uploaded dataset.”

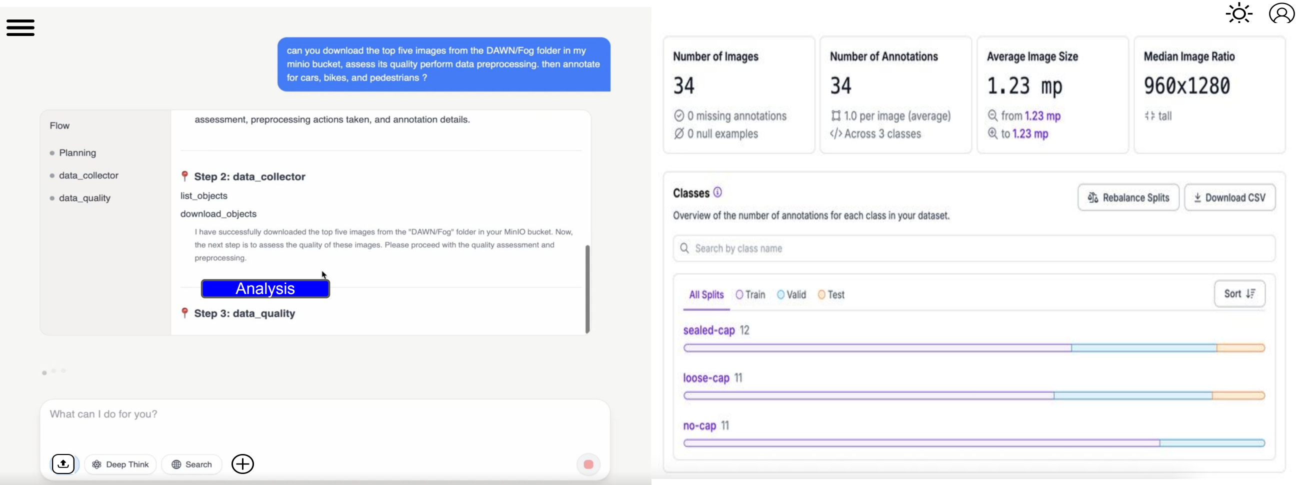
Example browser agent for automatic data collection



3.1 : **Execution Flows ~** Data Collection

System Response: Data analysis

- Split screen view :
  - The left panel shows the agent’s execution flow, along with the agent’s intermediate thoughts and “artefacts” (optional). The artefacts will maximize the right panel.
  - Right panel shows
    - Output artefacts : Only for visualizing the intermediate outputs of each step. Show output previews of 5 randomly selected images/videos
    - Interactable human in the loop components : To approve/deny/input the workflow intermediate outputs
- After data is collected, we could show metadata of the datasets depending on the availability of annotations.
- If annotations exist, we show class splits (as shown below)
- Further analysis could include “image dimensions”, “annotation heat map”, “Histogram count of objects by image”. Refer here for examples.



Source : roboflow

3.2 : **Execution Flows ~** Data quality

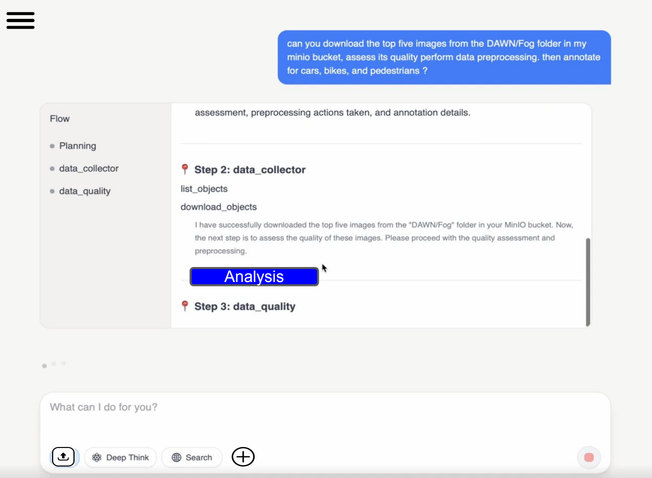
System Actions

- Fully autonomous
  - Open-ended request to perform quality analysis. Eg. “Analyse and detect degradations in the dataset”
  - No-reference IQA + VLM driven degradation detection. Refer here for more info.
- User specified
  - User specifies what type and metric to assess quality of images. Eg. “Detect hazy images and return a quality score using BRISQUE”
- Hybrid
  - User specifies what type and metric to assess quality of images.
  - But also asks agent to detect other degradations if any. Eg. “Detect hazy images and return a quality score using BRISQUE. Also detect if any other degradations exist.

3.2 : **Execution Flows ~** Data quality

System Response

- Display analysis on the data quality assessment :
    - Severity distribution : Count of severity by degradation type
    - Metric distribution : Box plots, histograms, violin plots, scatter plots by metric
- Further analysis could include “image dimensions”, “annotation heat map”, “Histogram count of objects by image”. Refer [here]() for examples.



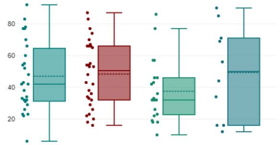
> High degradation
30 (10%)

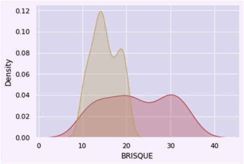
Avg BRISQUE
35

Avg Q-Align
3

Degradation Severity Distribution:

	Very Low	Low	Medium	High	Very High
Noise	0	4	1	0	0
Motion Blur	4	1	0	0	0
Defocus Blur	4	1	0	0	0
Haze	4	1	0	0	0
Rain	5	0	0	0	0
Dark	4	1	0	0	0
JPEG Artifacts	1	4	0	0	0





3.2 : **Execution Flows ~** Data preprocessor

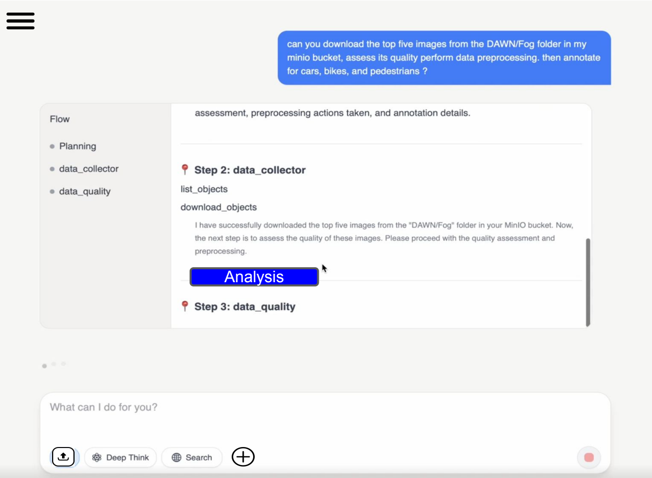
System Actions : Data preprocessor

- Fully autonomous
  - Open-ended request to perform preprocessing. Eg. “Preprocess the images”
  - The system should first perform autonomous data quality check.
  - Then divide the images into batches (by degradation type)
  - Then determine the preprocessing steps for each batch. Refer here for workflow.
- User specified
  - User specifies what type and metric to assess quality of images. Eg. “Detect hazy images and return a quality score using BRISQUE”
- Hybrid
  - User specifies what type and metric to assess quality of images.
  - But also asks agent to detect other degradations if any. Eg. “Detect hazy images and return a quality score using BRISQUE. Also detect if any other degradations exist.

3.2 : **Execution Flows ~** Data preprocessor

System Response: Data preprocessor

- Display analysis on before and after data processing
    - Stats showing % drop in degradation, BRISQUE scores, Q-Align scores
    - Severity distribution : Count of severity by degradation type
    - Metric distribution side by side comparison : Box plots, histograms, violin plots, scatter plots by metric



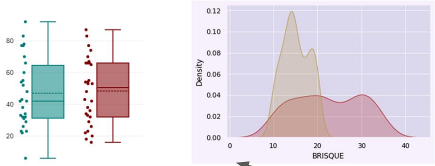
Degradation reduction 98%

Avg BRISQUE improvement 40%

Avg Q-Align improvement 50%

Degradation Severity Distribution:

	Very Low	Low	Medium	High	Very High
Noise	0	4	1	0	0
Motion Blur	4	1	0	0	0
Defocus Blur	4	1	0	0	0
Haze	4	1	0	0	0
Rain	5	0	0	0	0
Dark	4	1	0	0	0
JPEG Artifacts	1	4	0	0	0



Side by side comparison of quality metrics before and after processing

3.2 : **Execution Flows ~** Data Annotation

System Actions

- Fully autonomous
  -
- User specified
  -
- Hybrid
  -

3.2 : **Execution Flows ~** Data Annotation

System Response

3.2 : **Execution Flows ~** Data Augmentation

System Actions

- Fully autonomous
    -
- User specified
    -
- Hybrid
    -

3.2 : **Execution Flows ~** Data Augmentation

System Response

3.2 : **Execution Flows ~** Data Search

System Actions

- Fully autonomous
    -
- User specified
    -
- Hybrid
    -

3.2 : **Execution Flows ~** Data Search

System Response

Appendix

Agentic AutoML : A review

Table 1. Comparison between *AutoML-Agent* and existing LLM-based frameworks.

Framework	Key Functionality					
	Planning	**Verification**	**Full Pipeline**	**Task-Agnostic**	**Training-Free Search**	**With Retrieval**
AutoML-GPT (Zhang et al., 2023)	✗	✗	✗	✓	✓	✗
Prompt2Model (Viswanathan et al., 2023)	✗	✗	✓	✗	✗	✓
HuggingGPT (Shen et al., 2023)	✓	✗	✗	✓	✓	✓
CAAFE (Hollmann et al., 2023b)	✗	✓	✗	✗	✗	✗
MLCopilot (Zhang et al., 2024a)	✗	✗	✗	✓	✓	✗
AgentHPO (Liu et al., 2025)	✓	✓	✗	✓	✗	✗
Data Interpreter (Hong et al., 2024a)	✓	✓	✗	✓	✗	✗
DS-Agent (Guo et al., 2024a)	✓	✓	✗	✓	✗	✓
SELA (Chi et al., 2024)	✓	✓	✗	✓	✗	✗
Agent K (Grosnit et al., 2024)	✓	✓	✗	✓	✗	✓
AutoMMLab (Yang et al., 2025)	✗	✓	✓	✗	✗	✗
AutoML-Agent	✓	✓	✓	✓	✓	✓

AutoMMLab: Automatically Generating Deployable Models from Language Instructions for Computer Vision Tasks

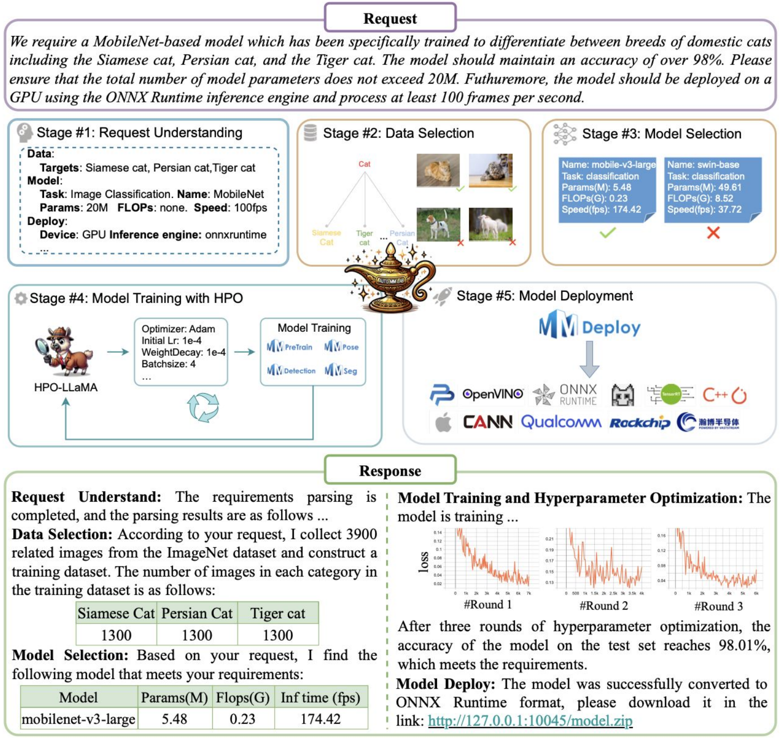


Figure 2: Overview of AutoMMLab. The workflow of AutoMMLab consists of five stages. **Request understanding:** Parse the language requests into formated configuration. **Data selection:** Select appropriate training data from the dataset zoo. **Model selection:** Select the optimal model from the model zoo. **Model training with HPO:** Train the model and optimize the hyper-parameters. **Model deployment:** Convert the model into a package compatible with the deployment environments.

The authors release a benchmark called LAMP for autoML tasks to evaluate :

1. Request Understanding :
   a. Key-level accuracy calculates the average accuracy of each key-value pair.
   b. Req-Level accuracy calculates the accuracy of understanding the entire request.
2. Hyperparameter Optimization : Propose LLM-based optimization (seems expensive to me, better to use Bayesian or other simpler methods that achieve similar performance)
3. End-to-end Evaluation : Pass/Fail

Hyperparameter optimization using LLaMA

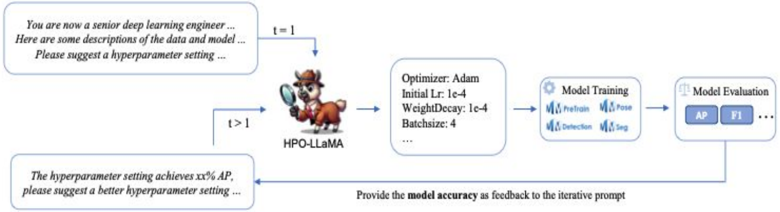


Figure 4: Overview of HPO-LLaMA. At the initial step ($t = 1$), HPO-LLaMA proposes a hyperparameter configuration based on the description of model and task. Model training is then executed and the training results are passed back to HPO-LLaMA via a text prompt for further rounds ($t > 1$).

AutoMMLab : Results

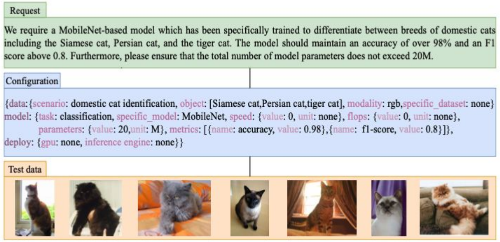
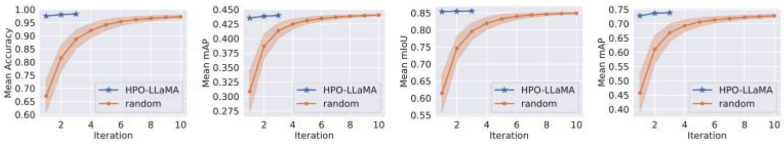


Figure 3: Example of LAMP dataset.

- RU-LLaMA finetuned on (request, configuration) pairs generated by GPT-4
- HPO-LLaMA finetuned on randomly selected 8000 (request, hyperparameters, performance) triplets.

Model	Key-Level			Req-Level
	Item	List	Total	
LLaMA2-7B-Chat	85.71	50.00	77.78	0
PaLM2	96.79	88.13	94.86	63.75
GPT-3.5-turbo	96.43	95.63	96.25	72.50
GPT-4	97.50	93.13	96.53	80.00
RU-LLaMA	**98.57**	**96.88**	**98.20**	**86.25**

Table 1: Evaluation of request understanding (RU). Best results are marked in bold.



(a) Image Classification (b) Object Detection (c) Semantic Segmentation (d) Keypoint Detection

Figure 5: HPO results of HPO-LLaMA and random sampling baselines on four tasks : (a) image classification, (b) object detection, (c) semantic segmentation and (d) keypoint detection. HPO-LLaMA demonstrates significantly higher efficiency.

Model	#R	Cls.	Det.	Seg.	Kpt.
BayesianRF	5	0.618±0.287	0.291±0.211	0.847±0.044	0.069±0.136
BayesianGP	5	0.761±0.264	0.280±0.208	0.848±0.041	0.081±0.150
LLaMA2-7B	1	0.839±0.213	0.128±0.164	0.291±0.409	0±0
PaLM2	1	0.964±0.056	0.367±0.196	0.845±0.067	0.719±0.079
GPT-3.5-turbo	1	0.849±0.214	0.364±0.194	0.852±0.044	0.204±0.160
GPT-4	1	0.861±0.188	0.434±0.147	0.803±0.194	0.096±0.158
HPO-LLaMA	1	<u>0.975±0.028</u>	<u>0.435±0.148</u>	<u>0.854±0.042</u>	<u>0.728±0.051</u>
HPO-LLaMA	3	**0.983±0.020**	**0.440±0.150**	**0.856±0.043**	**0.738±0.053**

Table 2: Evaluation of HPO. #R means the number of iteration rounds. The mean and standard deviation on for task are exhibited. Best results are marked in bold, second best results are underlined.

The models achieve similar results using random search, only slower (by 6-7 iterations). LLM optimization is an overkill and quite expensive if it doesn't converge.

RU	HPO	Cls.	Det.	Seg.	Kpt.	Total
LLaMA2-7B	LLaMA2-7B	0	0	0	0	0
PaLM2	PaLM2	14	25	27	15	81
GPT-3.5-turbo	GPT-3.5-turbo	24	24	25	11	84
GPT-4	GPT-4	17	27	29	14	87
RU-LLaMA	HPO-LLaMA	**31**	**31**	**32**	**18**	**112**

Table 3: End-to-end evaluation on LAMP benchmark. The assessment is based on a grading system scoring from '0' to '2', where '0' denotes 'total failure', '1' denotes 'workable model', 2 denotes 'perfect model'. The full score for each task is 40, and the total full score is 160.

SELA: TREE-SEARCH ENHANCED LLM AGENTS FOR AUTOMATED MACHINE LEARNING

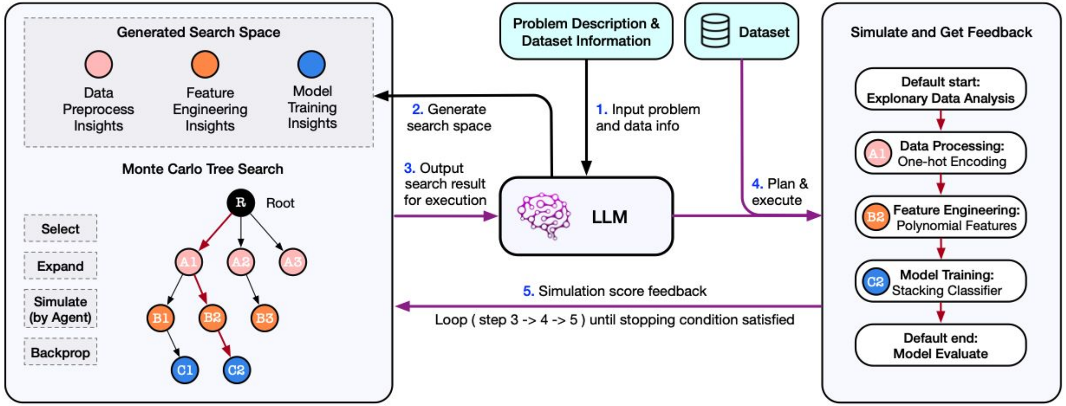


Figure 2: SELA's pipeline operates as follows: The system begins by inputting the problem description and dataset information into the LLM, which generates a search space of potential solutions, encompassing data preprocessing, feature engineering, and model training. The search module, powered by Monte Carlo Tree Search (MCTS), explores this space by selecting, expanding, and simulating potential configurations. The LLM agent then simulates the selected configuration by planning, coding, and executing the experiment. Feedback from the simulation is fed back into the search module, where it is used in the backpropagation step to refine future searches. This iterative process continues until a predefined stopping criterion is met, resulting in an optimized experimental pipeline.

RESULTS

$$NS(s_{\mathrm{raw}}) = \begin{cases} \frac{1}{1+\log(1+s_{\mathrm{raw}})} & \text{if the metric is RMSE.} \\ s_{\mathrm{raw}} & \text{otherwise.} \end{cases}$$

DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning

What is Case based reasoning ?

Retrieve similar past problems, reusing their solutions for the current problem, evaluating the effectiveness, revising the solution, and retaining successful solutions.

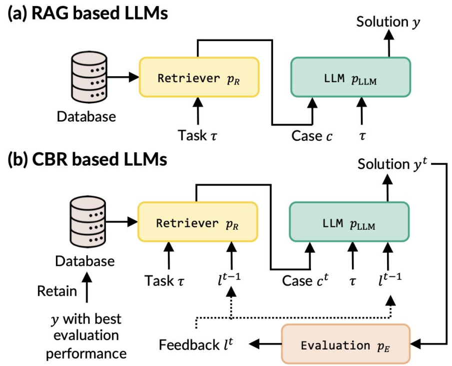


Figure 2. Comparison between (a) RAG based LLMs and (b) CBR based LLMs.

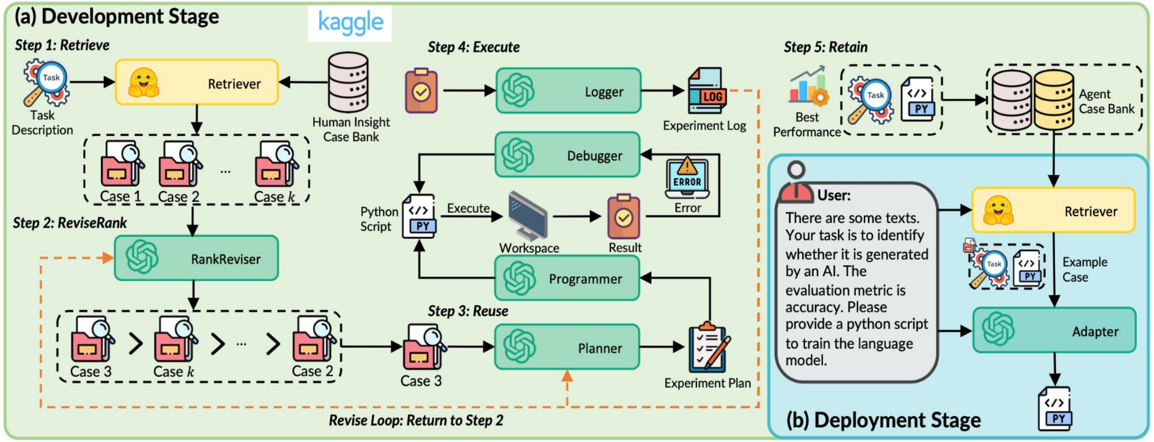


Figure 3. The diagram of DS-Agent. **(a) Development Stage:** DS-Agent structures an automatic iteration pipeline to build and revise the model based on execution feedback. **(b) Deployment Stage:** DS-Agent adapts past successful solutions for code generation.

DS-Agent: Results

Table 3. Mean rank w.r.t. task-specific evaluation metric results on 18 data science tasks in the deployment stage. Results are reported over 10 repetitive runs. Best performances are highlighted in bold, and second best performances are underlined.

		JS	HR	BPP	WR	DAG	BQ	TFC	WTH	ELE	SRC	UGL	HB	CA	CS	MH	SS	CO	SD	Avg
Mixtral -8x7b -Instruct	Zero-shot	37.0	35.0	35.0	31.0	35.0	32.0	29.0	32.0	30.0	44.0	54.0	46.0	73.1	66.6	65.8	63.6	33.7	72.0	45.3
	One-shot	35.2	35.0	32.2	31.0	35.0	29.1	29.0	32.0	30.0	36.5	47.1	46.0	50.1	53.1	51.2	51.1	23.6	61.5	39.4
	DS-Agent	37.0	35.0	35.0	31.0	35.0	32.0	29.0	32.0	30.0	20.1	16.4	38.5	25.3	54.5	53.7	53.9	32.2	47.6	35.5
GPT-3.5	Zero-shot	21.7	35.0	30.1	28.6	27.1	28.3	27.1	29.1	28.1	33.1	48.4	21.4	29.0	35.3	28.8	35.7	25.2	42.3	30.8
	One-shot	27.6	25.8	27.6	25.6	34.6	23.0	20.8	29.1	27.0	35.7	48.4	**21.1**	27.1	50.5	58.4	57.5	33.9	56.4	35.0
	DS-Agent	**6.0**	22.6	15.0	20.6	15.1	**13.1**	17.3	13.4	14.4	**20.0**	13.0	23.0	29.0	19.3	**7.6**	**2.0**	37.0	19.5	17.1
GPT-4	Zero-shot	36.7	31.8	35.0	29.0	29.4	32.0	29.0	32.0	30.0	37.3	45.7	33.6	**1.0**	**15.3**	23.2	17.9	28.3	20.1	28.2
	One-shot	35.1	24.4	**13.8**	26.6	29.6	28.8	23.1	30.1	26.6	26.7	41.6	36.7	29.7	21.9	35.3	28.9	21.4	23.2	28.0
	DS-Agent	18.6	**1.0**	14.6	**5.2**	**6.2**	18.8	**15.7**	**6.3**	**8.1**	**20.0**	**11.4**	21.2	**1.0**	32.6	14.5	8.2	**13.0**	**12.4**	**12.7**

Improves over past kaggle human experts

Table 2. Ablation results in terms of average best rank over 12 development tasks. Results are reported over five repetitive trials.

GPT-4	Average Best Rank
DS-Agent	**2.08**
DS-Agent w/o ReviseRank	2.58
DS-Agent w/o CBR	3.41

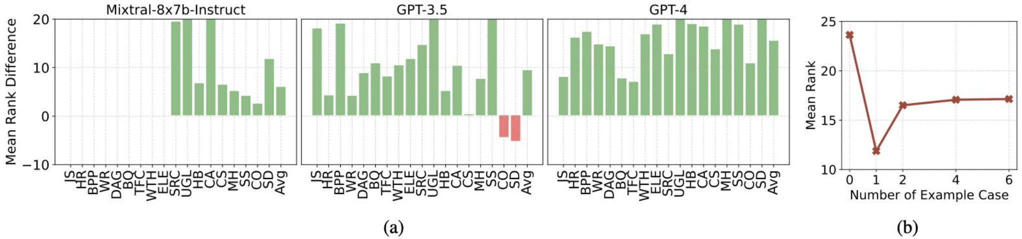
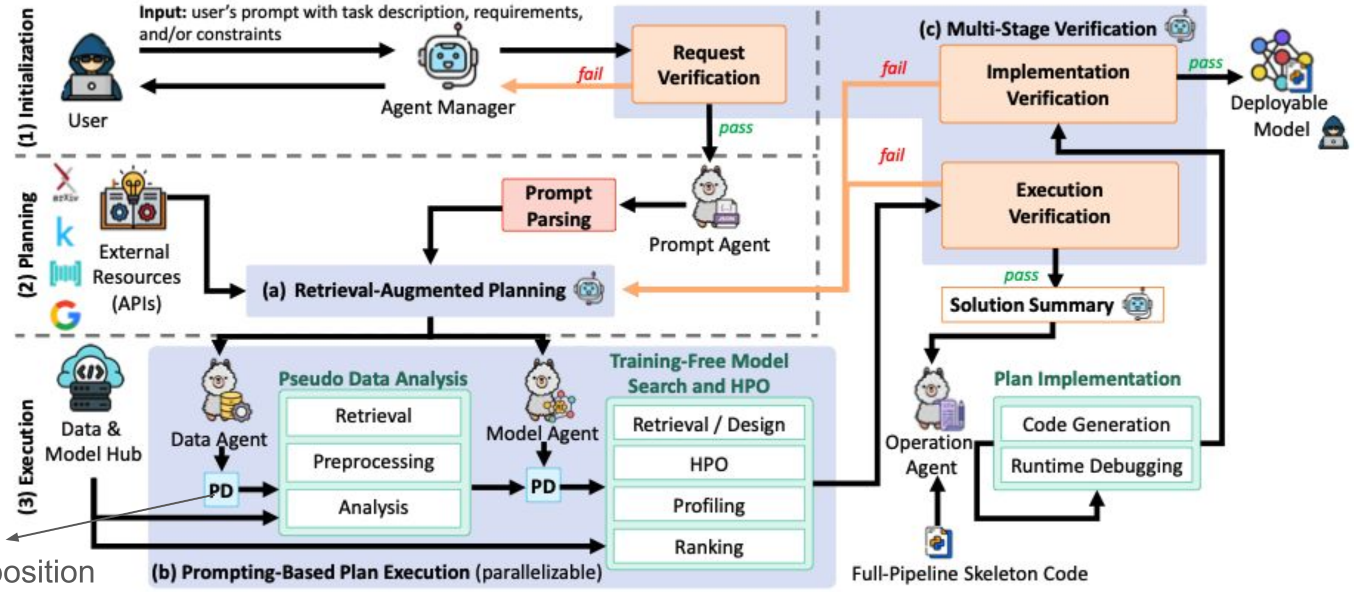


Figure 6. Further analyses on DS-Agent in the deployment stage. (a) Performance difference of DS-Agent learning from past successful experiences or textual human insights. (b) Hyper-parameter study on varying number of example case in DS-Agent with GPT-3.5.

AutoML-Agent: A Multi-Agent LLM Framework for Full-Pipeline AutoML



Plan decomposition

Figure 2. Overview of our *AutoML-Agent* framework. **(1) Initialization** stage aims to receive a valid user instruction using request verification. **(2) Planning** stage focuses on extracting ML related information by parsing the user instruction into a standardized form, and uses it to devise plans accordingly. **(3) Execution** stage executes each action given by the devised plans. Finally, based on the best execution results, *AutoML-Agent* outputs codes containing deployable model to the user.

AutoML-Agent : Results

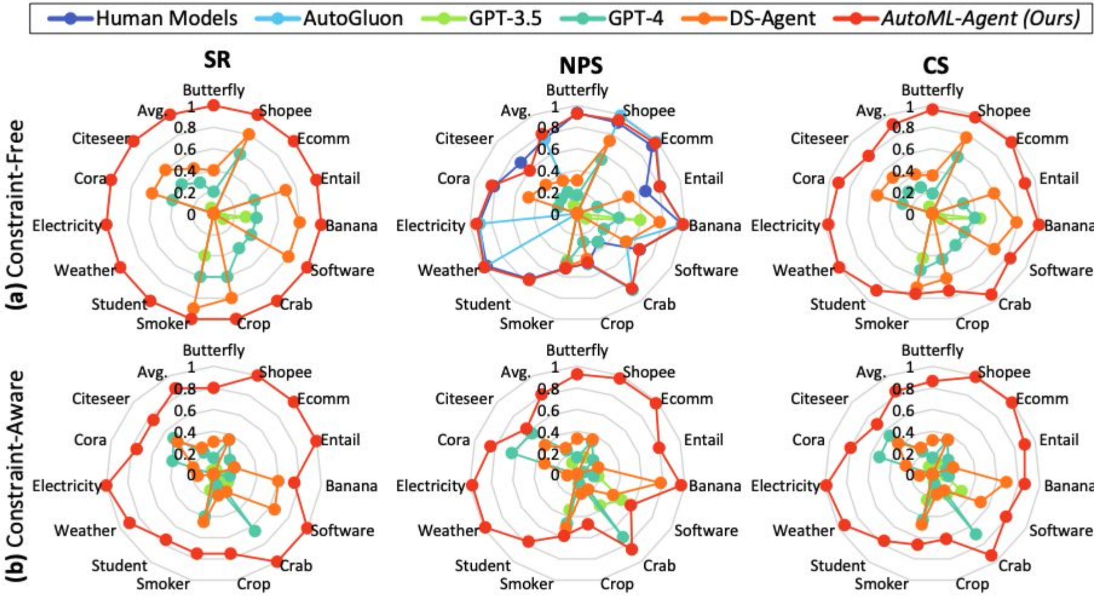


Figure 4. Performance comparison across all datasets using the SR, NPS, and CS metrics under (a) constraint-free and (b) constraint-aware settings. Higher scores indicate better results.

Figure 7. Average time and monetary cost breakdown.

Challenges of existing methods

Data challenges

- Lack of Automated data sourcing
- Lack of standardization of varied datasets (and annotation formats)
- Complex preprocessing and feature engineering for CV tasks
- Do not cover all CV tasks
- Skipped annotation, data augmentation entirely
- No data quality based verification step
- Preprocessing is fixed, does not depend on data distribution, quality and task

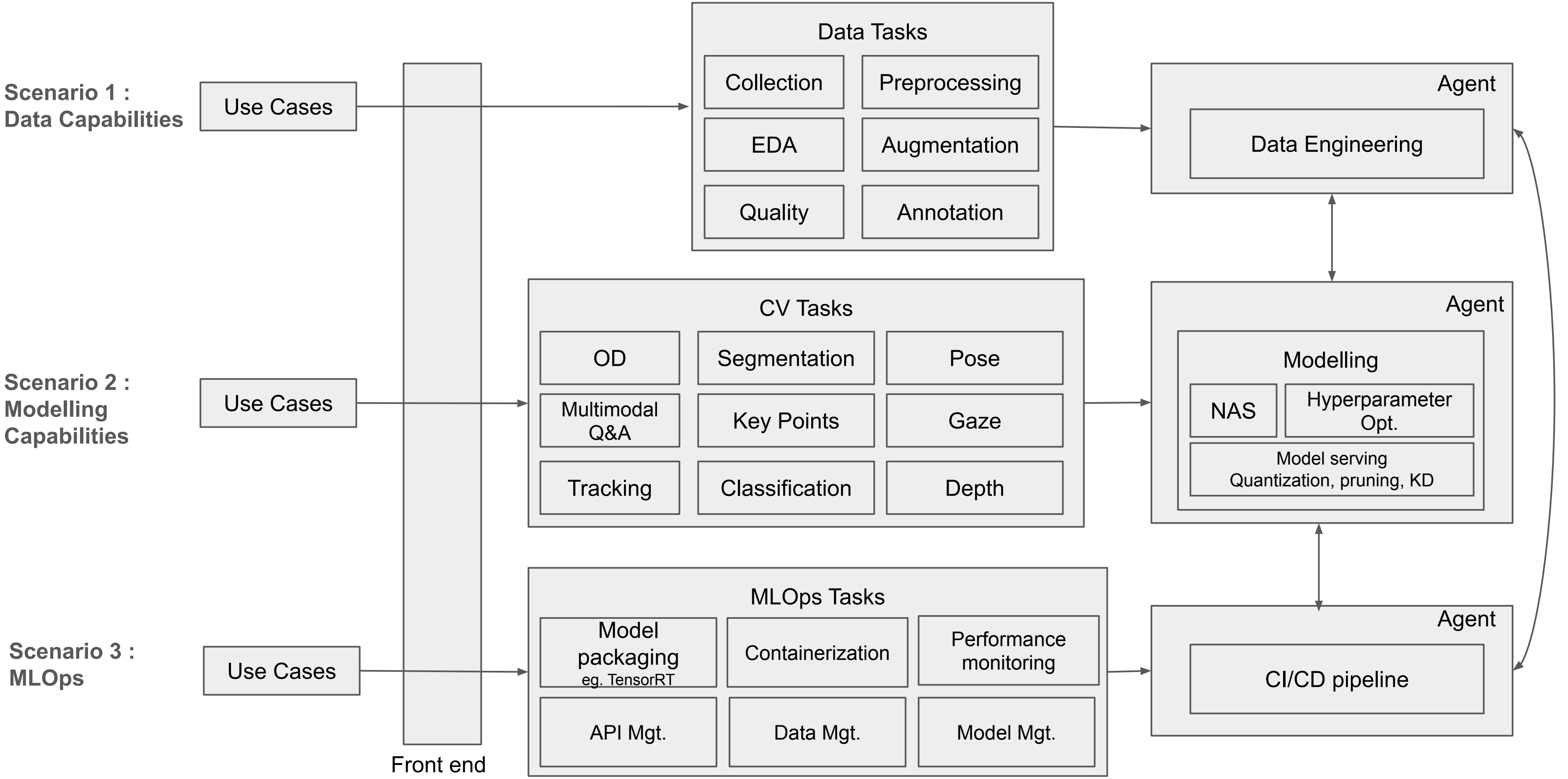
Model challenges

- HPO can get complex if model size and search space is high.
- HPO not applicable for foundation models. Might only need efficient PEFT.
- Cost vs Performance tradeoffs

Deployment challenges

- Don’t use proper logging for automatic error correction
- No human in the loop and/or unit testing of production level code
- Efficient serving using quantization, pruning and model conversion (ONNX Format) for different hardware backends
- Security and privacy risks, will require agent to run in a sandboxed environment (eg. docker)

Design Approach : How use cases trickle down to the tech stack



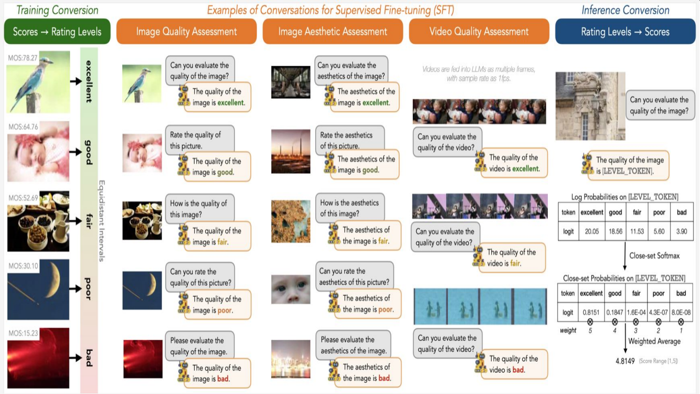
Quality assessment agent : NR-IQA

NR Method	Model names	Description
Q-Align	qalign (with quality[default], aesthetic options)	Large vision-language models
QualiCLIP(+)	qualiclip, qualiclip+, qualiclip+-clive, qualiclip+-flive, qualiclip+-spaq	QualiCLIP(+) with different datasets, koniq by default
LIQE	liqe, liqe_mix	CLIP based method
ARNIQA	arniqa, arniqa-live, arniqa-csiq, arniqa-tid, arniqa-kadid, arniqa-clive, arniqa-flive, arniqa-spaq	ARNIQA with different datasets, koniq by default
TOPIQ	topiq_nr, topiq_nr-flive, topiq_nr-spaq	TOPIQ with different datasets, koniq by default
TReS	tres, tres-flive	TReS with different datasets, koniq by default
FID	fid	Statistic distance between two datasets
CLIPIQA(+)	clipiqa, clipiqa+, clipiqa+_vitL14_512,clipiqa+_rn50_512	CLIPIQA(+) with different backbone, RN50 by default
MANIQA	maniqa, maniqa-kadid, maniqa-pipal	MUSIQ with different datasets, koniq by default
MUSIQ	musiq, musiq-spaq, musiq-paq2piq, musiq-ava	MUSIQ with different datasets, koniq by default
DBCNN	dbcnn	
PaQ-2-PiQ	paq2piq	
HyperIQA	hyperiqa	
NIMA	nima, nima-vgg16-ava	Aesthetic metric trained with AVA dataset
WaDIQaM	wadiqam_nr	
CNNIQA	cnniqa	
NRQM(Ma)[2]	nrqm	No backward
PI(Perceptual Index)	pi	No backward
BRISQUE	brisque, brisque_matlab	No backward
ILNIQE	ilniqe	No backward
NIQE	niqe, niqe_matlab	No backward
PIQE	piqe	No backward

Quality Assessment agent : Degradation classifier

Images

Q-Align [Github][paper]
Range = [1,5] (1 being bad, 5 being good)



BRISQUE
Range = [0,100] (0 being good, 100 being bad)

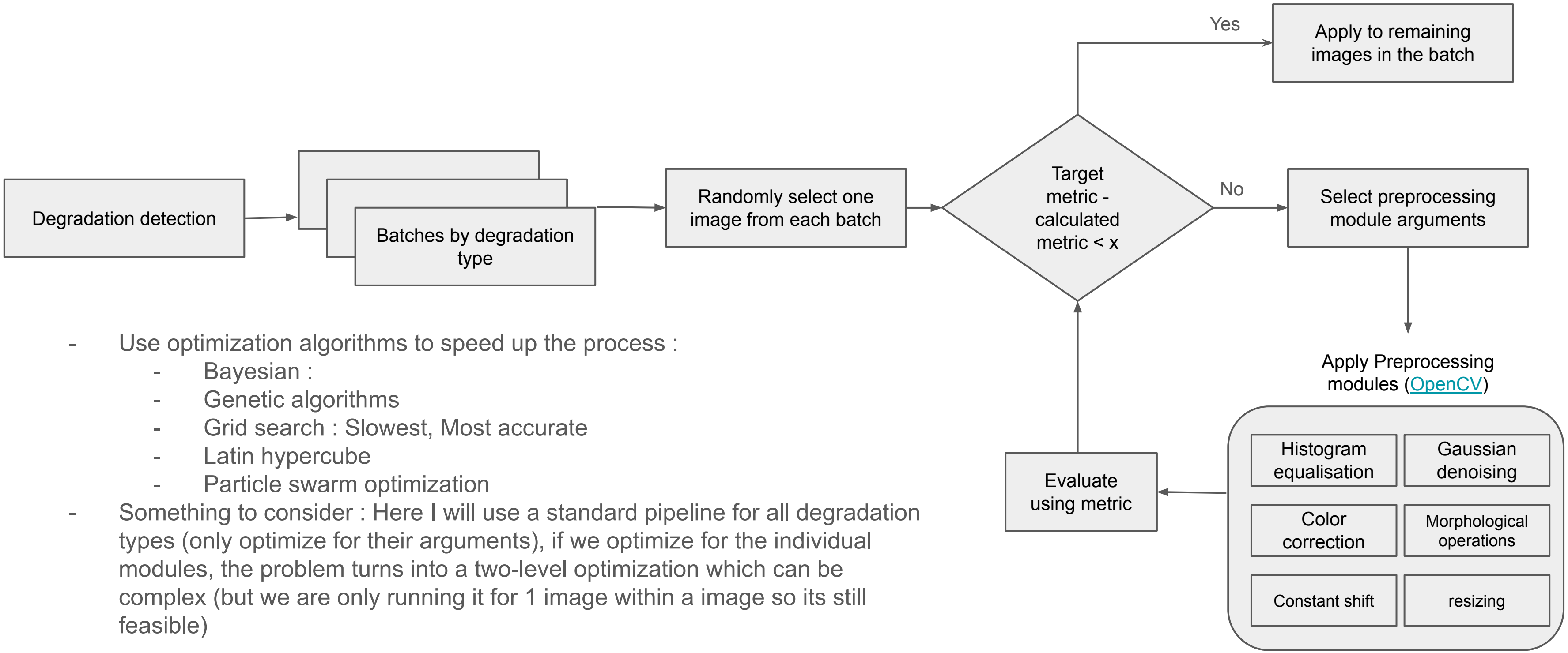
Gemini 1.5
Degradation classification

```
- Noise
- Motion Blur
- Defocus Blur
- Haze
- Rain
- Dark
- JPEG Compression Artifact

Each degradation is assigned one of five severity levels:
- Very Low
- Low
- Medium
- High
- Very High
```

Output :
degradation types
and severity levels

Data preprocessing - auto



- Use optimization algorithms to speed up the process :
    - Bayesian :
    - Genetic algorithms
    - Grid search : Slowest, Most accurate
    - Latin hypercube
    - Particle swarm optimization
- Something to consider : Here I will use a standard pipeline for all degradation types (only optimize for their arguments), if we optimize for the individual modules, the problem turns into a two-level optimization which can be complex (but we are only running it for 1 image within a image so its still feasible)