Agent for Numerical Data Retrieval and Understanding by Code Generation and Multimodal Reasoning

Florian Baud, Feda Almuhisen, Dorian Midou
CEA, IRFM, F-13108 Saint-Paul-lez-Durance, France
{florian.baud, feda.almuhisen, dorian.midou}@cea.fr

Abstract

Numerical data from sensors and time series are widely used in scientific research fields such as nuclear fusion experiments, which generate vast amounts of complex, high-dimensional data. Therefore, efficient numerical data analysis tools are crucial to accelerate experimental research. Large language models (LLMs) have emerged as promising solutions to analyze numerical data with natural language queries. However, LLMs have difficulties treating this type of data as they have been designed for text in the first place. To overcome these limitations, we propose a model-agnostic and data-agnostic agent that processes numerical data by code generation and multimodal reasoning. Our agent demonstrates competitive performance against baselines on benchmark data on numerical data tasks such as sensor data classification and time series understanding. While outperforming them on information retrieval benchmarks, also we have successfully applied our agent in the context of nuclear fusion research, where physicists and Tokamak operators interact with it to plan and analyze fusion experiments.

1 Introduction

Numerical data are ubiquitous in scientific research fields such as nuclear fusion, particularly when an experimental machine is equipped with numerous sensors producing vast amounts of data, which are analyzed to extract meaningful insights and prepare for more precise future or next experiments. For this reason, efficient exploration tools are critical for accelerating experiment preparation and advancing science. In nuclear fusion, physicists and Tokamak operators often rely on SQL and visual tools during the experimental campaign, requiring technical knowledge. This has motivated us to investigate a research assistant that is fast, robust, and easy to use for information retrieval and numerical understanding.

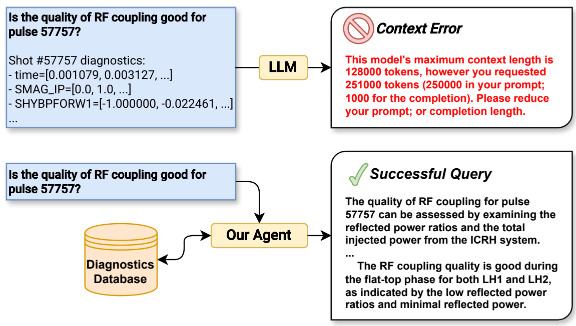


Figure 1: Processing multivariate numerical data as text fills the entire context windows. While our agent is able to process multivariate signals from a database.

The emergence of LLMs has paved the way for conversational assistants in research. One example is the Tokamak Copilot proposed in (Mehta et al., 2023), which integrates logbooks and text comments on tokamak operations. Extending the applicability of LLMs beyond text remains an active area of research. Various modalities such as vision (Liu et al., 2023; Bai et al., 2025) and audio (Chu et al., 2024) are well studied by the community. On the other hand, the numerical data modality remains under-explored. Whereas, their integration into LLMs could be beneficial for advancing scientific fields with vast amounts of numerical data.

The main issue is the tokenization of numerical data, as depicted in Figure 1. Depending on the strategy, tokenized numerical data become either a long sequence when encoded digit-by-digit (Spithourakis and Riedel, 2018) or lose precision when encoded by the standard BPE tokenizer (Sennrich et al., 2016). The split of numbers into individual digits is the standard strategy, as it can tokenize every number. It is noteworthy that the aggregation of digits is interpreted by the LLM, which could be misleading as shown by (Sivakumar and Moosavi, 2025).

However, dealing with long sequences of tokenized numerical data is challenging. To fit in the context window, custom methods often rely on win-

Findings of the Association for Computational Linguistics: ACL 2026, pages 38630–38653
July 2-7, 2026 ©2026 Association for Computational Linguistics

dowing (e.g., dividing the data into fixed or sliding segments), downsampling, or rounding to a lower number precision, which implies that the full information contained in the numerical data is truncated. We discarded these strategies as our goal is to create an assistant for scientific research where exactness counts.

Other methods represent numerical data by compressing it into an image (Yoon et al., 2024). This approach is effective for handing long sequences but lacks accuracy when it comes to retrieving information such as the maximum and minimum. On the contrary, a simple text-to-SQL method won't work for numerical understanding as it returns aggregated results only.

Inspired by the text-to-SQL method, we consider a method where the objective is to transform a textual query into a Python preprocessing script. The idea is to generate a preprocessing script that will produce relevant elements based on the text query. Then, an analysis step is necessary in order to process these elements and respond to the request. The advantage of this method is that it processes the results of the preprocessing script, which we require to be either reduced tables or images. Avoiding processing the numerical data in its entirety.

Under these settings, our contributions are as follows:

- We propose an agent for handling numerical data by generating code and reasoning on multimodal outputs. It combines numerical data as text and images where both modalities are obtained via code generation.
- We provide a benchmark that compares our agent against text-only and image-only methods between open-source models on the tasks of sensor data understanding, time series understanding, and information retrieval.
- We have created an aggregated dataset comprising various sensor data types, including human activity, electrocardiogram, and respiration data, along with general time series data. This dataset, available on Hugging Face, serves to enhance understanding and information retrieval in sensor and time series data.

2 Related work

This work focuses on handling numerical data in conversational settings. Below, we review existing approaches that discuss the integration of this modality with LLMs.

Text-to-SQL methods While not directly addressing continuous sensor data, have inspired the ability of LLMs to interact and query structured databases. Benchmarks like Spider (Yu et al., 2018), introduced a large-scale, cross-domain benchmark with 10,181 questions over 200 databases, designed to evaluate the generalization to unseen schema and SQL queries. Similarly, BIRD (Li et al., 2023) provides over 12,751 unique question-SQL pairs, and a benchmark of 95 databases, with a particular emphasis on evaluating the impact of extensive database content on text-to-SQL parsing. Spider 2.0 (Lei et al., 2025) introduces a comprehensive benchmark specifically dedicated for real-world enterprise scenarios. Despite significant advancements in text-to-SQL for information retrieval, such as Spider and BIRD, these methods assume structured, tabular data and cannot effectively process long sequence continuous sensor data, to support analytical tasks without preprocessing.

LLM and MLLM methods Recognizing the limitations of text-to-SQL methods for numerical data, researchers have investigated the capacity of LLMs to process and interpret continuous data. (Fons et al., 2024) evaluate LLMs on time series feature understanding using only text modality. They introduce a comprehensive taxonomy and benchmark, highlighting strengths and weaknesses of LLMs in interpreting various time series characteristics. Authors in (Yoon et al., 2024) propose a visual prompting method for sensor data using multimodal LLMs (MLLMs) to interpret sensor information. They also introduce a visualisation generator tool that creates optimal visualisatisions for specific sensor tasks without the need for task-specific prior knowledge. SensorLLM (Li et al., 2025) aligns sensor inputs with automatically generated trend descriptions to address the lack of semantic context in time series data. The authors propose a two stage framework that maps sensor sequences to natural language using special tokens, allowing effective human activity recognition.

Agentic methods In addition to direct processing of continuous sensor data by LLMs, a parallel research efforts investigates the development of intelligent agents coupled with LLMs to automate data science workflows. For instance, (Huang et al., 2024) present MLAgentBench a benchmark

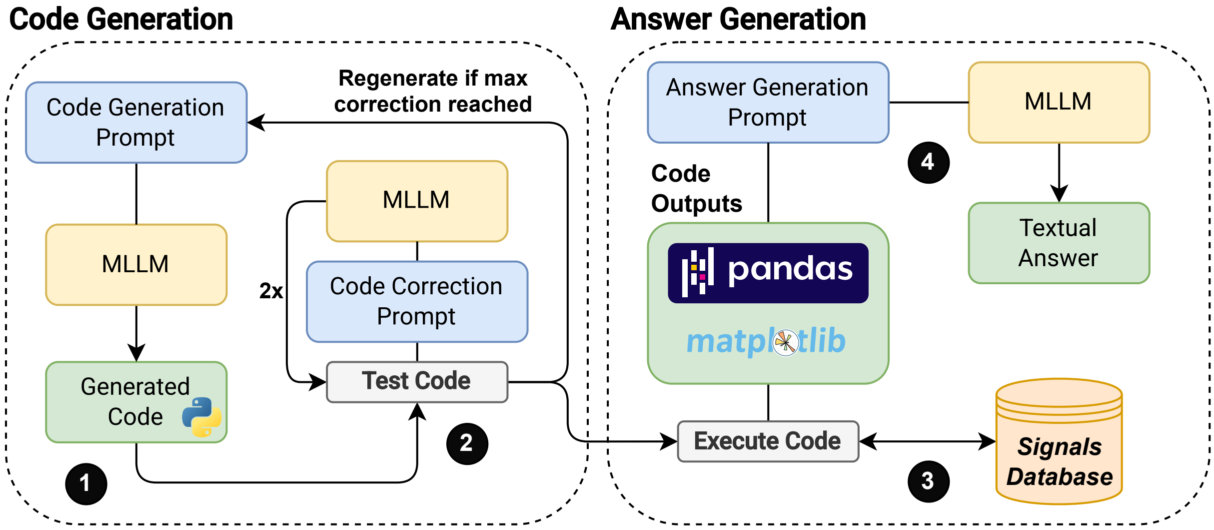


Figure 2: Overview of the proposed agent. For an incoming request, (1) the agent starts by generating a Python code (2) that is debugged if the code execution goes wrong. We have limited the number of debugs loop to two attempts. After that, the code is regenerated from the beginning. (3) Then, the agent executes the code with numerical data, result is multimodal outputs. (4) These are integrated in the answer generation prompt and the model generates an answer.

designed to evaluate LM based agents capacity in performing a machine learning experimentation. Their work includes prompting based LM agent. Furthermore, (You et al., 2025) propose a notebook-centric LLM agent framework, that allows users to perform data analysis and visualisation through natural language queries. This framework automates code generating and execution within a notebook environment. However, DatawiseAgent primarily focuses on automating data science tasks such as data exploration, visualisation, and supports training and evaluation of machine learning models for structured data, relying on notebook-centric approach. (Yang et al., 2024) propose MatPlotBench a Text-to-Visualization benchmark that is designed for scientific data visualisation. While, (Nathani et al., 2025) introduce MLGYM and MLGym-Bench a new framework and benchmark for evaluating and developing agents and models for AI research task.

Most of existing solutions require extensive preprocessing steps to handle particular types of sensor data. The proposed agent addresses these limitation by directly operating on raw long-sequence sensor data and eliminate the necessity of a specialized preprocessing pipelines. This is achieved by dynamically generating code tailored to the specific user query and data type.

The proposed agent is able to debug, diagnose, and correct errors in generated code, which allows a more robust automated workflow for processing any type of sensor data.

3 Numerical Data Agent

The goal of our agent is to answer requests about sensor data and time-series data. It should handles classification (deriving conclusions about sensor data), information retrieval (retrieving min/max values and mean in a span of time), and analysis tasks (correlation, trend, etc.). To address these tasks, the agent must leverage multiple modalities.

For example, the classification of a signal performs better when using a graph representation, as shown by (Yoon et al., 2024). On the other hand, retrieving an exact value from a graph representation leads to an approximate value. In this case, it is better to treat numerical data as text until the context window can handle it. As there is no single modality which could cover all requests, the agent should figure out which modalities are relevant given the user request.

However, an agent that merely redirects requests to image or text prompts is not scalable to a vast amount of data. As a result, all the data points will be encoded into the routed modality, exceeding the context length in the text modality case. For instance, in nuclear fusion, each experiment creates around 6,600 different signals that are sampled at 2 ms. This means that the data needs to be filtered before it can be routed. We were inspired by the text-to-SQL method to let the agent choose which filter is suitable. Although this method seemed promising, it was devoid of preprocessing tools.

The advancement of models' coding capabili-

{system_prompt}

Goal: Python Code Generation for Sensor Data Processing

Your task is to generate syntactically correct and executable Python code within a single markdown block. DO NOT answer the question directly.

Output Requirements

1. Objects: Generate code to create up to {num} useful objects. These must be either Matplotlib figure objects (matplotlib.pyplot.Figure) or Pandas DataFrames (pandas.DataFrame).
2. Return: All generated objects must be appended to the predefined list variable result.
3. Focus: The objects must be directly relevant to the user's question, preparing for subsequent analysis.
   - Visualizations (Matplotlib figures) are often suitable for classification, trend, or distribution analysis.
   - DataFrames (Pandas) are often suitable for statistical summaries, raw values, or intermediate feature tables.
4. No Model Training: DO NOT train a machine learning or deep learning model. The code's purpose is data preparation and visualization for subsequent analysis.
5. Size Constraint: Be cautious about not returning excessively large Pandas DataFrames; the code should return only necessary or aggregated elements. For example, returning raw Pandas Dataframes is not a good practice.

The generated Python code must include all necessary steps to process, transform, filter, and/or aggregate the data in df before creating the final output objects.

Context and Available Data

- Data Description: Descriptions about the data acquisition process: {data_description}
- DataFrame Schema: The structure of the Pandas DataFrame: {table_info}
- The DataFrame is loaded into the variable df.

Strict Execution Rules

- Starting Snippet (MUST USE AS IS):

```python
# Available packages/tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import statsmodels.api as sm
import scipy
import ruptures as rpt
import pywt

result = [] # Initialize plots or pandas dataframes list
df = pd.read_pickle("./data.pkl") # Load sensor data
```

- Column Names: Use only the column names visible in the provided schema. Do not query for non-existent columns.
- Data Integrity: Do not create, simulate, or modify the underlying data.
- Visualization Rules: Do not save plots to disk. Avoid using plt.subplot or multi-plot axes; each figure must be a single plot.
- Final Output: The resulting code must be in a single markdown block for execution.

Figure 3: System prompt for code generation

ties allows us to generate scripts that filter and preprocess data dynamically. Our agent generates a Python code that will create either a pandas DataFrame or a Matplotlib graph. It determines which filter, preprocessing, and modalities the request requires in a single call. As illustrated in Figure 2, the workflow of the proposed agent is divided into two main stages: code generation and answer generation.

The code generation prompt 3 in (1) includes details such as maximum number of multimodal outputs, the description and the schema of the sensor data. At this stage, the model determines which modality (plot or dataframe) is necessary for answering the question. The number of multimodal outputs of the code generation step is determined by the `num` parameter as shown in the prompt. The description is specific for each sensor data type and provides information such as the acquisition process, the sampling rate, and the context. The schema gives an overview of the data, including the type of each column, a sample of the data, and the shape. In this way, the model is fully informed about the data that it will manipulate.

To guide the model in generating an accurate code, we have included a starting code to the prompt, as shown in figure 3. The model must follow this starting code for two reasons (i) the variable `result` stores the multimodal outputs which we gather; and (ii) the variable `df` is a placeholder for coherence with the previously declared schema and instructions. It also includes a list of available packages of the Python environment, which can be extended at will.

We found that sometimes the generated code does not run because it contains errors. In this case, the erroneous code is passed to a debugging loop until the code runs. It can be fully regenerated if the debugging loop fails above an arbitrary number of attempts.

When the code is successfully executed, the outputs contained in the result variable are checked. They must be either pandas dataframes or matplotlib plots and must not exceed a fixed number.

The second call to the MLLM is made for answering the question given the outputs of the generated code. As shown in Figure 4, the prompt includes the data description, the generated code, the response format, and the obtained pandas dataframes in Markdown format.

```
You are an expert system designed to answer questions about
preprocessed sensor data. Your primary directive is to provide
answers based exclusively on the context provided below. Do
not use any external knowledge or make assumptions beyond
what is stated in the data description and preprocessing
script. Outputs obtained from the preprocessing script
are either pandas DataFrames or images. You will find
them in the context.{data_description}{generated_code}
{response_format}{text_pandas}
Question: {user_input}
```

Figure 4: Generation prompt

The proposed agent is model-agnostic, yet the underlying language model should respect some conditions. Firstly, the model should accepts multimodal inputs, text and image. Secondly, it must have capabilities in code generation. It should be noted that a specialized code model could be used for the first step. In our case we investigated the same model for code and answer generation as we intend to deploy our assistant on low ressources scenario. For benchmarking purpose, we investigated a hybrid setting where the code is generated by a specialized model.

Our agent facilitates the processing of long-sequence and multivariate numerical data for sensor and time series understandings and provides precise values for information retrieval tasks. The agent responses can be explained by investigating the generated code and the obtained outputs from it.

Hybrid Configuration This agent's workflow can operate with a model specialized in code generation. By replacing the first workflow call to the specialized model. This configuration aims to mitigate the bottleneck caused by the code. Indeed, the outputs generated by the code have a direct impact on the response of the second call to the MLLM model. In addition, this configuration allows models that do not have code generation capabilities to be evaluated.

4 Experiments

To prove the capabilities of our agent, we have aggregated sensor data from (Yoon et al., 2024) and time series from (Fons et al., 2024) into a single dataset. It is composed of classification tasks that are translated in to a question-answer format. For each classification dataset, we create one question that corresponds to the classification task. This approach aims at evaluating reasoning on sensor data and time series. The information retrieval task is conducted on time series data. In this section, we present the proposed benchmark, followed by the experimental setup and a discussion of our results.

4.1 Benchmark Data

In this study, we address sensor data classification, information retrieval, and time series understanding tasks. A brief description of the data is provided below.

The JP-Morgan time series dataset (Fons et al., 2024) includes a range of synthetic univariate and multivariate time series, each annotated with corresponding textual descriptions. This benchmark is designed to evaluate an LLMs ability to interpret and analyze time series features and patterns. The univariate time series include characteristics such as trends (directional movements over time, up and down), seasonality patterns that repeat over a fixed or irregular period (fixed period constant amplitude,etc), volatility which is the degree of dispersion of a series over time (constant increasing, clustered, leverage effect), outliers or significant deviations from typical patterns (spikes, step-spikes, level shifts, temporal disruptions), structural breaks (shifts in the series data such as regime changes or parameter shifts), and statistical properties (like fat tails, and stationarity versus non stationarity). The multivariate time series capture interdependencies between variables, such as cross correlation (the relationship between two series at different time lags). The dataset supports multiple task types feature detection, classification, value retrieval and arithmetic reasoning.

Data from ByMyEyes (Yoon et al., 2024), this dataset includes real world sensor data paired with visual prompts and textual task descriptions, designed to evaluate MLLMs in reasoning over four sensor modalities: accelerometer, electrocardiography (ECG), electromyography (EMG), and respiration sensor. It integrates data from multiple sources, including HAR (Stisen et al., 2015) for human activity recognition (running and walking), UTD-MHAD (Chen et al., 2015) for fine-grained arm motion classification, and Swimming (Brunner et al., 2019) for swimming style recognition. ECG-based arrhythmia diagnosis detection tasks are derived from PTB-XL (Wagner et al., 2020), covering four symptom classes. Gesture tasks include hand gesture recognition from (Ozdemir et al., 2022), while respiration-based stress detection tasks utilize the WESAD dataset (Schmidt et al., 2018).

We use data from JP-Morgan for time series understanding and information retrieval (specifically for arithmetic data) and ByMyEyes data for sensor data understanding. The benchmark data are

Model	HAR	UTD -MHAD	Swimming	PTB -XL	Gesture	WESAD	Time Series	Total
Text-only Method				F1 Score				
Mistral Small	9.3	2.8	4.6	11.4	0.0	0.0	39.0	9.6
Gemma 3	15.5	2.2	14.2	14.5	0.0	0.0	40.5	12.4
Qwen 2.5	11.5	3.1	22.2	8.3	0.0	0.0	**49.0**	13.4
Image-only Method								
Mistral Small	10.3	1.6	13.0	6.9	5.4	46.8	36.4	17.2
Gemma 3	18.1	0.7	15.2	7.6	**6.3**	32.2	29.6	15.7
Qwen 2.5	17.6	1.4	11.9	7.6	4.8	**55.7**	39.4	**19.8**
Numerical Data Agent								
Mistral-Small	14.8	3.5	18.5	**15.4**	5.5	39.1	41.9	**19.8**
Gemma 3	7.2	2.0	19.9	10.7	5.0	21.8	34.0	14.4
Qwen 2.5	13.4	2.7	19.3	15.3	4.2	22.3	41.7	17.0
Hybrid Numerical Data Agent								
Qwen 2.5 Coder + Llava 1.6	**19.9**	2.1	**35.9**	11.7	5.9	21.1	33.4	18.6
Qwen 3 Coder + Llava 1.6	19.4	3.6	33.1	12.9	4.7	21.4	34.9	18.6
Qwen 3 Coder + Mistral Small	12.2	3.2	16.9	13.8	5.7	40.4	37.1	18.5
Devstral 2505 + Mistral Small	11.9	**4.2**	21.7	11.6	5.8	17.2	38.1	15.8

Table 1: F1 score performance on sensor data understanding and time series understanding (numerical data understanding).

summarized in Table 4 in the appendix.

4.2 Setups

Our experimental setup is similar to ByMyEyes (Yoon et al., 2024), where non-expert users analyse sensor data using MLLMs. However, the preprocessing in ByMyEyes follows a fixed pipeline which limits the adaptability to new sensor data. In this work, we assume that the preprocessing should be done by the agent itself for adaptability. In addition, we included time series understandings and information retrieval which are not covered by ByMyEyes. We also target deployment on private networks where data confidentiality is required. Given the computational constraint for running our agent in such a manner, we benchmark small MLLMs that run on two A40 GPUs using vLLM (Kwon et al., 2023) for model serving on private network.

We evaluated our agent with Qwen2.5-VL-32B-Instruct (Bai et al., 2025), Mistral-Small-3.2-24B-Instruct-2506 and Gemma-3-27b-it (Team et al., 2025). These models are small enough to be deployed in a private network and show strong capabilities in instructions following, code generation, and visual reasoning. We additionally evaluate our agent in a hybrid configurations where code generation step is handled by a code-specialized model and the generation step by a MLLM or VLM. For this scenario, we evaluated llava-v1.6-mistral-7b-hf (Liu et al., 2024) with Qwen2.5-Coder-32B-Instruct (Hui et al., 2024) and Mistral-Small-3.1-24B-Instruct-2503 with Devstral-Small-2505.

We compare our agent with and without hybrid settings against two baselines: a text-only method and an image-only method. The first one processes numerical data as text format, which may be limited by the model's context window. While, the second encodes sensor and time series data as straightforward matplotlib plot of value versus time. We refer to these four methods as the *Text-only Method*, *Image-only Method*, *Numerical Data Agent*, and *Hybrid Numerical Data Agent*.

All benchmarked method includes the zero chain of thoughts (CoT) (Kojima et al., 2022) prompting in their system prompt to enhance reasoning. For our agent, we set the `num` parameter of the code generation prompt to the number of signals present in the evaluated task plus an arbitrary value, restricting the multimodal outputs.

To reduce the computational cost, we cache the result of the code generation step. This allows the

Model	Min Value		Max Value		Value on Date		Min Date	Max Date	Total
Text-only Method	EM	MAPE	EM	MAPE	EM	MAPE	EM		
Mistral Small	90.0	0.0	95.5	0.0	35.0	0.451	43.0	38.0	60.3
Gemma 3	72.5	0.0	77.0	0.06	32.0	0.483	36.5	27.5	49.1
Qwen 2.5	91.5	0.0	94.5	0.033	35.5	0.718	44.5	31.0	59.4
Image-only Method									
Mistral Small	13.0	0.267	15.0	0.09	2.5	1.09	6.0	11.5	9.6
Gemma 3	15.0	0.365	13.5	0.217	3.0	0.883	2.5	4.0	7.6
Qwen 2.5	15.5	0.336	17.0	0.073	1.5	0.961	5.0	9.5	9.7
Numerical Data Agent									
Mistral Small	94.0	0.0	99.0	0.0	53.5	0.013	88.5	88.5	84.7
Gemma 3	**94.5**	0.0	**99.5**	0.0	39.5	0.384	72.0	72.0	75.5
Qwen 2.5	93.5	0.006	99.0	0.11	92.0	0.213	97.5	98.5	96.1
Hybrid Numerical Data Agent									
Qwen 2.5 Coder + Llava 1.6	84.5	0.0	89.0	0.0	80.5	0.021	88.0	88.0	86.0
Qwen 3 Coder + Llava 1.6	82.5	0.0	86.5	0.0	82.5	0.0	87.0	87.0	85.1
Qwen 3 Coder + Mistral Small	93.5	0.0	98.5	0.0	**95.0**	0.0	**99.0**	**99.0**	**97.0**
Devstral 2505 + Mistral Small	**94.5**	0.0	**99.5**	0.0	47.0	0.038	84.5	84.0	81.9

Table 2: This table reports the exact match (EM) and the mean absolute percentage error (MAPE) for information retrieval benchmark. Unparsed predictions are taken into consideration for scores calculation.

generated code to stay identical during the entire inference of a task, thereby incapacitating the variety of generated codes. For diversifying them, we run the benchmark with 10 workers involving 10 different codes for the same task.

4.3 Results

We organize our results around two complementary tasks: numerical data understanding, and information retrieval. The sensor data understanding task has long sensor data, whereas time series understanding task has short length data. We assess the four previously mentioned methods on both short and long numerical data to cover these two scenarios. Also, we analyze modality selection regarding the models used by our agent. Furthermore, we plotted polar graphs to investigate each function used by task type in the agent generated code and compare the F1 score. You can find them in appendix D. Our experimental evaluation is across 16 tasks and 6 datasets as summarized in Appendix B Table 4.

Numerical Data Understanding Table 1 presents the overall F1-Score for each method on the sensor data and time series understanding tasks. Text-only approaches can handle short-length numerical data, but they fail to handle long-length data such as Gesture and WESAD, where their F1-score is zero, due to context window overflow (see 4 summary for dataset size details).

Comparatively, the image-only approach achieves the highest F1-score on the Gesture and WESAD datasets (55.7% for Qwen 2.5 on WESAD, 6.3% for Gemma3 on Gesture), demonstrating that image modality is particularly well-suited for long-length numerical data. We hypothesize that plotting long sequences applies implicit compression that preserves task relevant patterns while filtering high-frequency noise.

Our agent demonstrates competitive performance against both baselines in single model and hybrid configurations. We observe gains on PTB-XL against the image-only method across all models. This demonstrates that code-based preprocessing can extract relevant ECG features that visual inspection alone may miss. As it was shown in Appendix D for the PTB-XL polar plots.

For time series understanding task, our agent showed a competitive results against the text only method. However, the correlation task results in Appendix Table 3 demonstrate that the agent outperforms all competing methods. This superior performance is primarily attributed to the agent's ability to calculate correlation coefficients. This is particularly evident in Appendix D Figure 8, where the Mistral Small model calculates the Pearson coefficient using the *scipy.stats.pearsonr* function and obtains an F1 score of 0.8. Nevertheless, the Figure 10 for Qwen shows that he only used the *plt.subplots* function, but if we look carefully at the code generated in Figure 6, we see that he calculates the correlation coefficient with pandas. We observe the same behavior for Gemma 3 on the generated code in Figure 7.

Additionally, Figure 8 illustrates that Mistral Small computes the slope of the time series to ascertain the trend. However, when it calculates the slope with the *sm.OLS* function, the F1 score is lower than when analyzing the time series plot with the *plt.plot* function. We observed similar behaviour in most of the models. Where we concluded that the model interpretation can mislead the agent performance. This is due to the fact that models misinterpret the trend by looking only at the slope sign, which misleads it when there is no trend.

Our agent is greatly influenced by code generation, as shown in the correlation task. Still, we note that the agent is competitive on short-length numerical data, often performs better on medium-length data, but fails on long data sequences. We observe that the image-only method benefits from compression for long sequences. Where this compression is not always applied to our agent due to the variability of the generated code, as shown in the figures of appendix D. On the other hand, the agent is better when the task requires precise computation, like correlation or information retrieval.

Information Retrieval The information retrieval task aims to extract values from the time series: the maximum and the date, the minimum and the date, and the value at a specific date. Unlike understanding task, we do not cache the generated code because the requested value on a date varies for each example of the dataset. Results are presented in Table 2.

Our agent outperforms both baselines on information retrieval tasks. Qwen 2.5 achieves 96.1% total exact match (EM) compared to 59.4% for text-only and 9.7% for image-only. The agent generates a code that searches for the exact value in the time series rather than inferring them from visual or textual representations. This is reflected in the MAPE scores: the agent returns values directly from the data, achieving near zero error (e.g., 0.013 for Mistral Small) compared to 0.451-1.09 for baseline methods. As shown in Figure 5, all models correctly generate pandas DataFrames for the retrieval task, demonstrating appropriate modality selection. However, some models still produce unnecessary figures alongside the required DataFrames. But, it is noteworthy that Mistral has the lowest level of figure generation, even though it does not achieve the best performance in information retrieval. In this specific case, Mistral's coding capabilities are limited for information retrieval, as it achieves the best performance when combined with the Qwen 3 coding model.

Modality Selection Analysis Figure 5 shows that models differ in output modality selection. Mistral Small generates figures 70-80% of the time on numerical data tasks, while Gemma3 generates figures only 40-50%, preferring DataFrames. This correlates with performance: on HAR, Mistral-Small achieves 14.8% F1 versus Gemma 3's 7.2%. However, for retrieval tasks, all models appropriately generate DataFrames (figure proportion below 20%), explaining consistent strong performance across configurations.

This analysis allows us to derive guidelines for every model to reduce the model's preferences. We recommend relying on in-context learning depending on chosen outputs by the model. Models where figures are preferred on arithmetic or information retrieval tasks should integrate in-context examples during code generation. On the contrary, models that rely on tables for classification tasks should have examples that drive the model to generate figures.

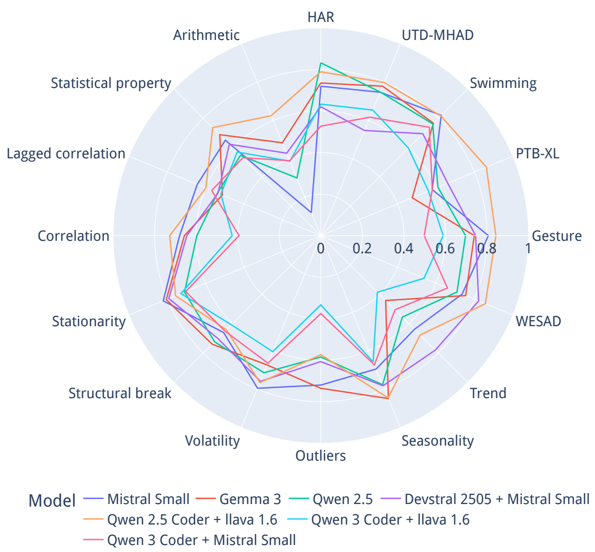


Figure 5: Proportion of figure in multimodal outputs for each model and numerical data tasks.

WEST Tokamak Application To evaluate the utility in practical scientific context, we deployed our proposed agent to process continuous sensor data from fusion experiments conducted on the WEST Tokamak (Bucalossi et al., 2024). The agent operates as an end-to-end question answering assistant with Mistral-Small as MLLM. We present responses on examples for information retrieval and data analysis tasks in Appendix E. We can provide more examples for the WEST Tokamak question-answer application. We provide explicit examples in Appendix E.

5 Conclusion

We propose an agent for sensor data understanding, time series understanding, and information retrieval. The agent leverages the coding and vision capabilities of small open-source MLLMs to process long-length and multivariate numerical data. In the perspective to locally deploy this agent for assisting researchers in the nuclear fusion field. We evaluated our agent on the proposed benchmark, where the agent shows competitive performance against two baseline methods. However, the performance is highly influenced by the backbone model that generate the code. Indeed, we found that the coding capability is critical for our agent, especially for data science code generation. We also tested a hybrid agent with specialised code model, and the benchmark results has slightly improved the performance. Our perspective is to improve this benchmark by working in specialised data science model to investigate further the effect of preprocessing.

Acknowledgements

This work was granted access to the HPC resources of IDRIS under the allocation 2025-AD011016343 made by GENCI. This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Limitation

Our agent demonstrates a robustness for treating numerical data understanding tasks and information retrieval. Nevertheless, it can face some limitations regarding the requested query, as the code generation could not cover all possible cases.

This is due to the straightforward nature of our agent, which consists of a two-step pipeline designed to minimize computational costs. For this reason, we did not explore advanced methods for assessing the generated code, which could avoid utilizing unnecessary functions for processing. Furthermore, the agent's models were not jointly optimized in order to preserve the model-agnostic capability of our agent.

Similarly, we do not evaluate large MLLMs, as our priority is to set our agent production-ready on a private network. Nevertheless, it could be interesting to explore the performance of our agent using large MLLMs.

We do not integrate advanced chain-of-thoughts strategies in our agent as we only used zero-shot CoT. Notably, in the code generation prompt which is essential for our agent to answer the request as shown by the Gemma3 model outperforming other model on statistical property detection task. A future work could integrate this strategy to enhance the code generation capability of our agent, especially to show reasoning examples for preprocessing the data.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Gino Brunner, Darya Melnyk, Birkir Sigfússon, and Roger Wattenhofer. 2019. Swimming style recognition and lap counting using a smartwatch and deep learning. In *Proceedings of the 2019 ACM International Symposium on Wearable Computers*, ISWC '19, page 23–31, New York, NY, USA. Association for Computing Machinery.

J. Bucalossi, A. Ekedahl, , the WEST Team, J. Achard, K. Afonin, O. Agullo, T. Alarcon, L. Allegretti, F. Almuhisen, H. Ancher, G. Antar, Y. Anquetin, S. Antusch, V. Anzallo, C. Arnas, J.F. Artaud, M.H. Aumeunier, S.G. Baek, and 373 others. 2024. West full tungsten operation with an iter grade divertor. *Nuclear Fusion*, 64(11):112022.

Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. 2015. Utd-mhad: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 168–172.

Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, Junyang Lin, Chang Zhou, and Jingren Zhou. 2024. Qwen2-audio technical report. *Preprint*, arXiv:2407.10759.

Elizabeth Fons, Rachneet Kaur, Soham Palande, Zhen Zeng, Tucker Balch, Manuela Veloso, and Svitlana Vyetrenko. 2024. Evaluating Large Language Models on Time Series Feature Understanding: A Comprehensive Taxonomy and Benchmark. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21598–21634, Miami, Florida, USA. Association for Computational Linguistics.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. Mlagentbench: Evaluating language agents on machine learning experimentation. *Preprint*, arXiv:2310.03302.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, volume 35, pages 22199–22213. Curran Associates, Inc.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, pages 611–626, New York, NY, USA. Association for Computing Machinery.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Preprint*, arXiv:2305.03111.

Zechen Li, Shohreh Deldari, Linyao Chen, Hao Xue, and Flora D. Salim. 2025. SensorLLM: Aligning large language models with motion sensors for human activity recognition. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 354–379, Suzhou, China. Association for Computational Linguistics.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2024. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 26296–26306.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Viraj Mehta, Joseph Abbate, Allen Wang, Andrew Rothstein, Ian Char, Jeff Schneider, Egemen Kolemen, Cristina Rea, and Darren Garnier. 2023. Towards LLMs as operational copilots for fusion reactors. In *NeurIPS 2023 AI for Science Workshop*.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. 2025. MLGym: A New Framework and Benchmark for Advancing AI Research Agents. *arXiv preprint*. ArXiv:2502.14499 [cs].

Mehmet Akif Ozdemir, Deniz Hande Kisa, Onan Guren, and Aydin Akan. 2022. Dataset for multi-channel surface electromyography (semg) signals of hand gestures. *Data in Brief*, 41:107921.

Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger, and Kristof Van Laerhoven. 2018. Introducing wesad, a multimodal dataset for wearable stress and affect detection. In *Proceedings of the 20th ACM international conference on multimodal interaction*, pages 400–408.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Jasivan Alex Sivakumar and Nafise Sadat Moosavi. 2025. How to Leverage Digit Embeddings to Represent Numbers? In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 7685–7697, Abu Dhabi, UAE. Association for Computational Linguistics.

Georgios Spithourakis and Sebastian Riedel. 2018. Numeracy for language models: Evaluating and improving their ability to predict numbers. In *Proceedings* *of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2104–2115, Melbourne, Australia. Association for Computational Linguistics.

Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne, and Mads Møller Jensen. 2015. Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, SenSys '15, page 127–140, New York, NY, USA. Association for Computing Machinery.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *Preprint*, arXiv:2503.19786.

Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. 2020. Ptb-xl, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15.

Zhiyu Yang, Zihan Zhou, Shuo Wang, Xin Cong, Xu Han, Yukun Yan, Zhenghao Liu, Zhixing Tan, Pengyuan Liu, Dong Yu, Zhiyuan Liu, Xiaodong Shi, and Maosong Sun. 2024. MatPlotAgent: Method and Evaluation for LLM-Based Agentic Scientific Data Visualization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11789–11804, Bangkok, Thailand. Association for Computational Linguistics.

Hyungjun Yoon, Biniyam Aschalew Tolera, Taesik Gong, Kimin Lee, and Sung-Ju Lee. 2024. By My Eyes: Grounding Multimodal Large Language Models with Sensor Data via Visual Prompting. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2219–2241, Miami, Florida, USA. Association for Computational Linguistics.

Ziming You, Yumiao Zhang, Dexuan Xu, Yiwei Lou, Yandong Yan, Wei Wang, Huaming Zhang, and Yu Huang. 2025. DatawiseAgent: A Notebook-Centric LLM Agent Framework for Automated Data Science. *arXiv preprint*. ArXiv:2503.07044 [cs].

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

A Time Series Results

Model	Trend	Seas.	Out.	Vol.	Struct. break	Stat.	Cor.	Lagged cor.	Total
Text-only Method					F1 Score				
Mistral Small	80.1	**38.2**	24.8	4.8	44.0	35.3	53.4	**33.2**	39.0
Gemma 3	95.5	31.1	**69.2**	6.7	44.0	12.3	60.4	30.0	40.5
Qwen 2.5	**98.0**	35.0	63.6	9.7	**48.5**	43.0	62.6	32.5	**49.0**
Image-only Method									
Mistral Small	97.5	33.8	41.6	4.6	37.0	43.4	40.3	29.5	36.4
Gemma 3	79.2	30.4	40.5	7.9	28.5	10.1	45.4	24.3	29.6
Qwen 2.5	97.5	31.1	41.8	**17.3**	34.0	**44.4**	54.1	28.5	39.4
Numerical Data Agent									
Mistral-Small	77.7	35.6	46.0	17.2	16.8	30.7	75.3	20.2	41.9
Gemma 3	22.5	30.1	43.2	10.5	22.8	6.4	78.9	17.3	34.0
Qwen 2.5	66.3	31.3	41.7	13.2	28.8	10.7	**83.8**	18.4	41.7
Hybrid Numerical Data Agent									
Qwen 2.5 Coder + Llava 1.6	46.6	21.8	28.2	14.1	28.9	28.7	58.6	18.7	33.4
Qwen 3 Coder + Llava 1.6	60.4	18.2	29.4	10.7	27.7	21.0	65.1	13.5	34.9
Qwen 3 Coder + Mistral Small	62.2	34.7	34.2	5.4	41.2	12.2	54.6	25.7	37.1
Devstral 2505 + Mistral Small	69.5	35.5	43.1	3.7	22.6	18.5	65.9	15.7	38.1

Table 3: Results for each time series task. (Seas. for Seasonality, Out. for Outliers, Vol. for Volatility, Struct. break for Structural break, Stat. for Stationarity, Cor. for Correlation and Lagged cor. for Lagged correlation)

B Datasets Summary

Name	Label Count	Feature Count	Example Count	Sequence Length Mean	Sequence Length Median	Sequence Length Min	Sequence Length Max	Numerical Precision
Sensor Data								
HAR	6	4	335	9 013	2 634	101	78 049	9
UTD-MHAD	21	7	670	178	176	107	326	18
Swimming	5	12	2 262	813	808	49	6 382	16
PTB-XL	5	2	7 500	1 000	1 000	1 000	1 000	5
Gesture	10	5	2 000	12 000	12 000	12 000	12 000	20
WESAD	3	2	45	515 697	464 800	253 400	838 600	15
Time Series								
Trend	3	2	200	90	90	60	125	18
Seasonality	4	2	200	68	60	24	125	18
Outliers	4	2	200	77	90	30	125	18
Volatility	5	2	200	134	120	90	200	18
Structural Break	3	2	200	75	60	30	125	18
Stationarity	5	2	200	78	90	30	125	18
Correlation	3	3	200	56	60	30	90	18
Lagged Correlation	3	3	200	59	60	30	90	18
Statistical Property	2	2	200	80	90	30	125	18
Arithmetic	200	2	200	84	90	24	200	18

Table 4: Dataset summary

C Generated Code

```python
# Available packages/tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

result = []  # Initialize plots or pandas dataframes list
df = pd.read_pickle("./data.pkl")  # Load sensor data

# Step 1: Compute the Pearson correlation coefficient
correlation_matrix = df[['Series_1', 'Series_2']].corr()
correlation_coefficient = correlation_matrix.loc['Series_1', 'Series_2']
correlation_summary = pd.DataFrame({
    'Variable': ['Series_1 vs Series_2'],
    'Correlation Coefficient': [correlation_coefficient],
    'Correlation Type': ['Positive' if correlation_coefficient > 0 else 'Negative' if
    ↪  correlation_coefficient < 0 else 'None']
})
result.append(correlation_summary)

# Step 2: Visualize the relationship with a scatter plot
fig_scatter, ax_scatter = plt.subplots(figsize=(8, 6))
ax_scatter.scatter(df['Series_1'], df['Series_2'], color='blue', alpha=0.6)
ax_scatter.set_title('Scatter Plot of Series_1 vs Series_2')
ax_scatter.set_xlabel('Series_1')
ax_scatter.set_ylabel('Series_2')
ax_scatter.grid(True)
result.append(fig_scatter)

# Step 3: Plot both series over time
fig_time_series, ax_time_series = plt.subplots(figsize=(12, 6))
ax_time_series.plot(df['Date'], df['Series_1'], label='Series_1', color='red')
ax_time_series.plot(df['Date'], df['Series_2'], label='Series_2', color='green')
ax_time_series.set_title('Time Series Plot of Series_1 and Series_2')
ax_time_series.set_xlabel('Date')
ax_time_series.set_ylabel('Values')
ax_time_series.legend()
ax_time_series.grid(True)
result.append(fig_time_series)

# Final Output: The result list contains the correlation summary and visualizations
```

Figure 6: Generated code from Qwen 2.5 using our agent on the Correlation time series task.

```python
# Available packages/tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn as sk
import statsmodels.api as sm
import scipy
import ruptures as rpt
import pywt

result = [] # Initialize plots or pandas dataframes list
df = pd.read_pickle("./data.pkl") # Load sensor data

# Calculate the Pearson correlation coefficient
correlation = df['Series_1'].corr(df['Series_2'])

# Create a DataFrame to store the correlation
correlation_df = pd.DataFrame({'Correlation': [correlation]})
result.append(correlation_df)

# Visualize the time series
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], df['Series_1'], label='Series_1')
plt.plot(df['Date'], df['Series_2'], label='Series_2')
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Time Series Plot')
plt.legend()
result.append(plt.gcf())

# Calculate rolling correlation
rolling_window = 10
rolling_corr = df['Series_1'].rolling(window=rolling_window).corr(df['Series_2'])

# Visualize rolling correlation
plt.figure(figsize=(10, 6))
plt.plot(df['Date'], rolling_corr, label='Rolling Correlation')
plt.xlabel('Date')
plt.ylabel('Correlation')
plt.title(f'Rolling Correlation ({rolling_window} window)')
plt.legend()
result.append(plt.gcf())
```

Figure 7: Generated code from Gemma 3 using our agent on the Correlation time series task.

D Polar plots of function versus F1 score

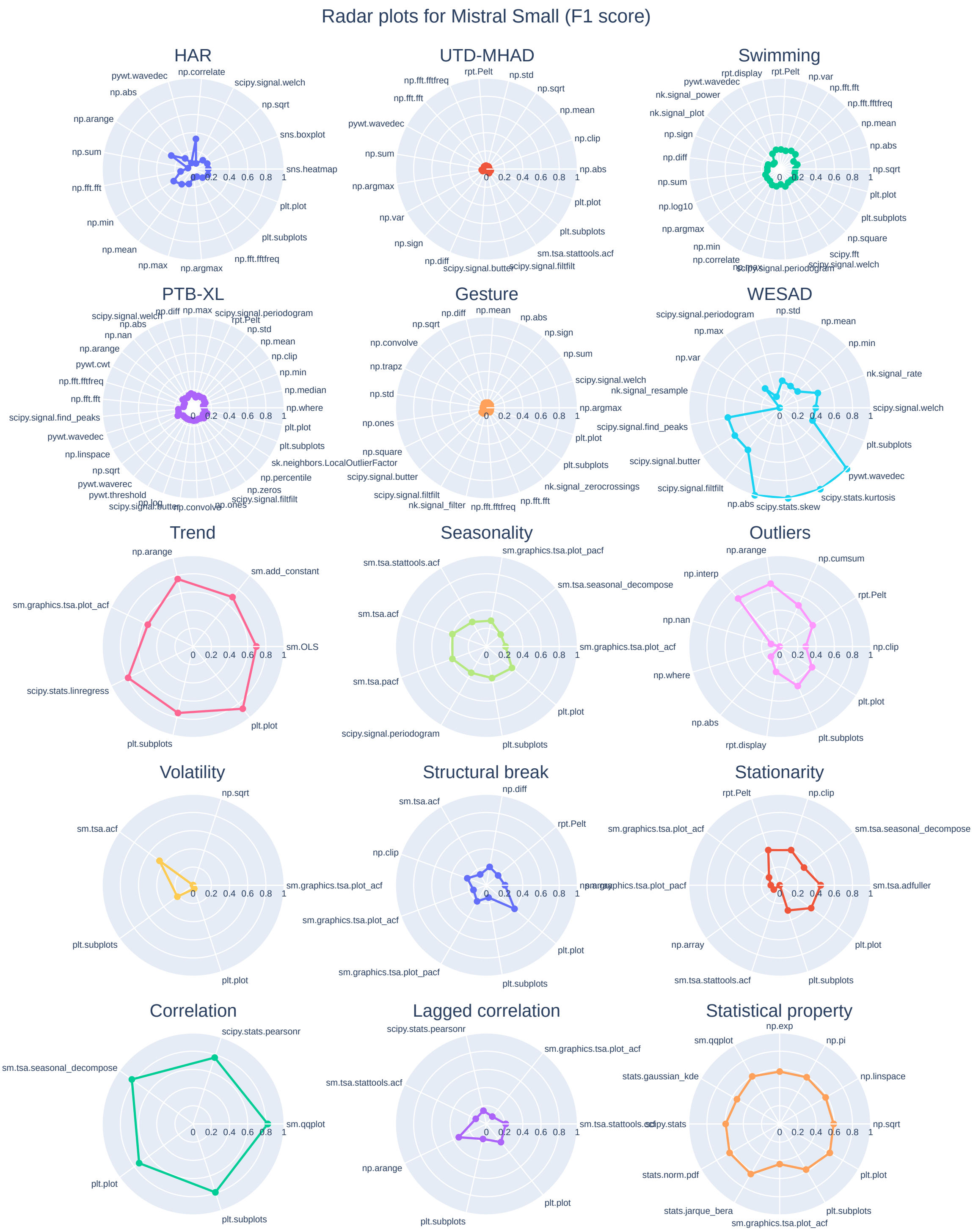


Figure 8: Mistral Small

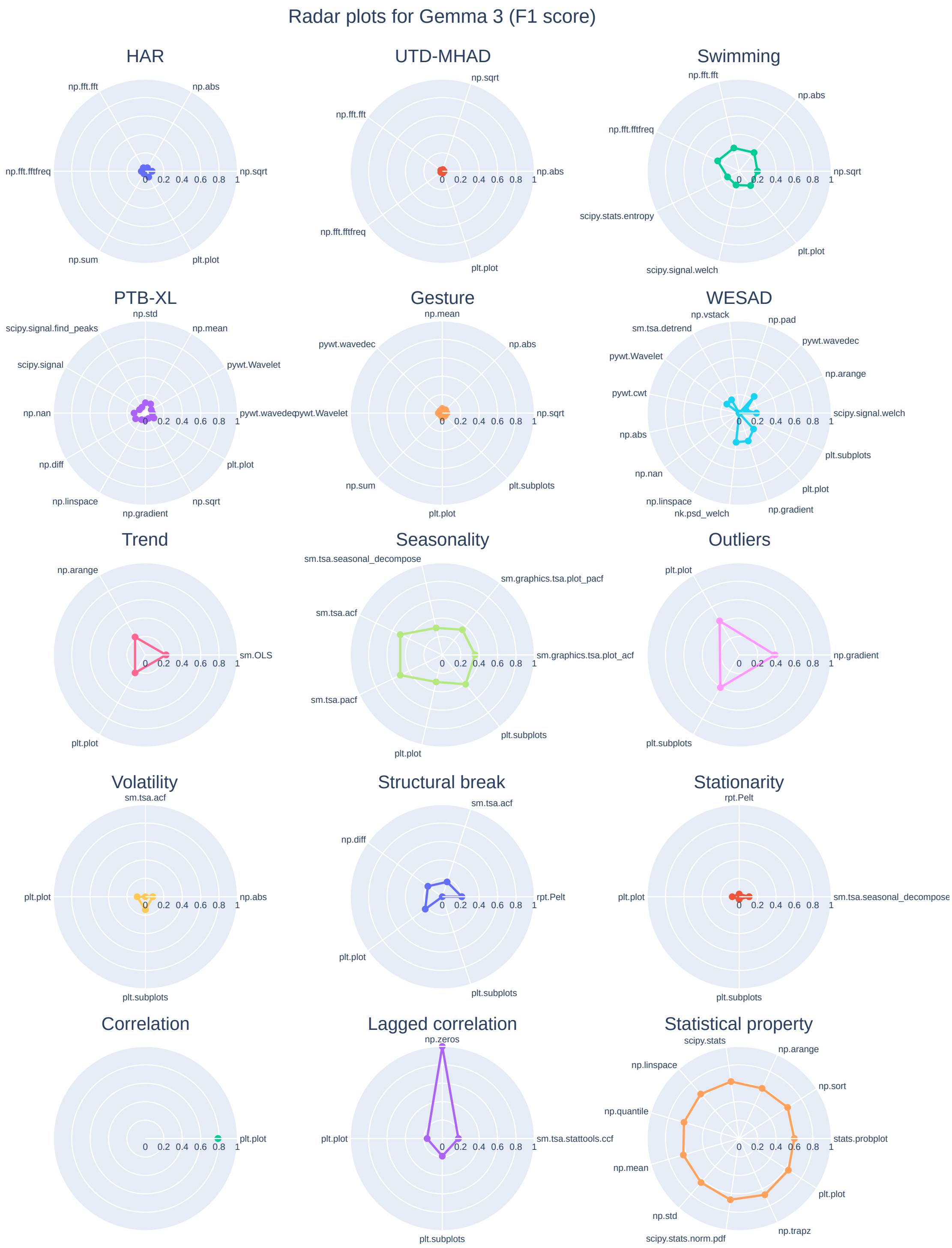


Figure 9: Gemma 3

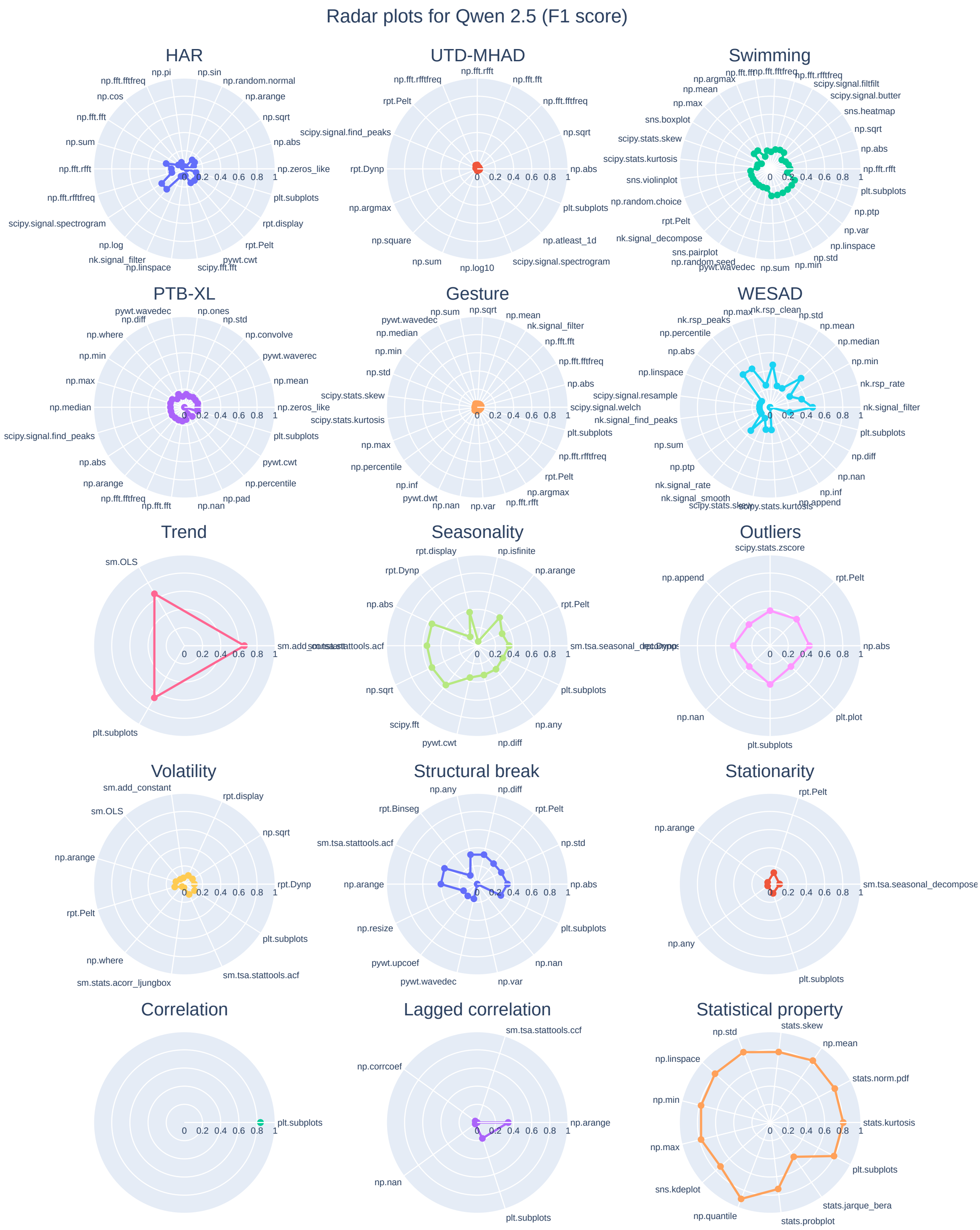


Figure 10: Qwen 2.5

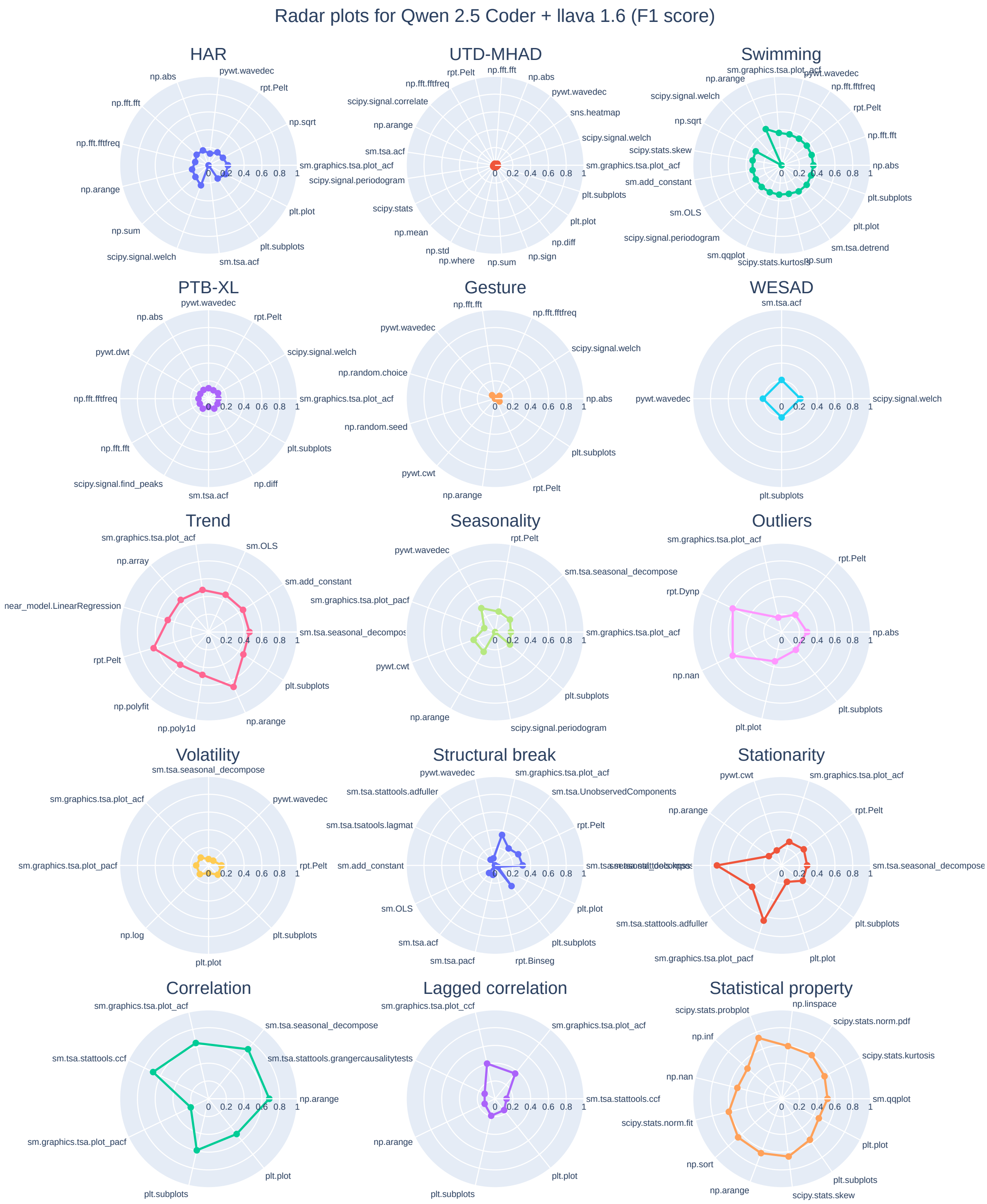


Figure 11: Qwen 2.5 Coder + llava 1.6

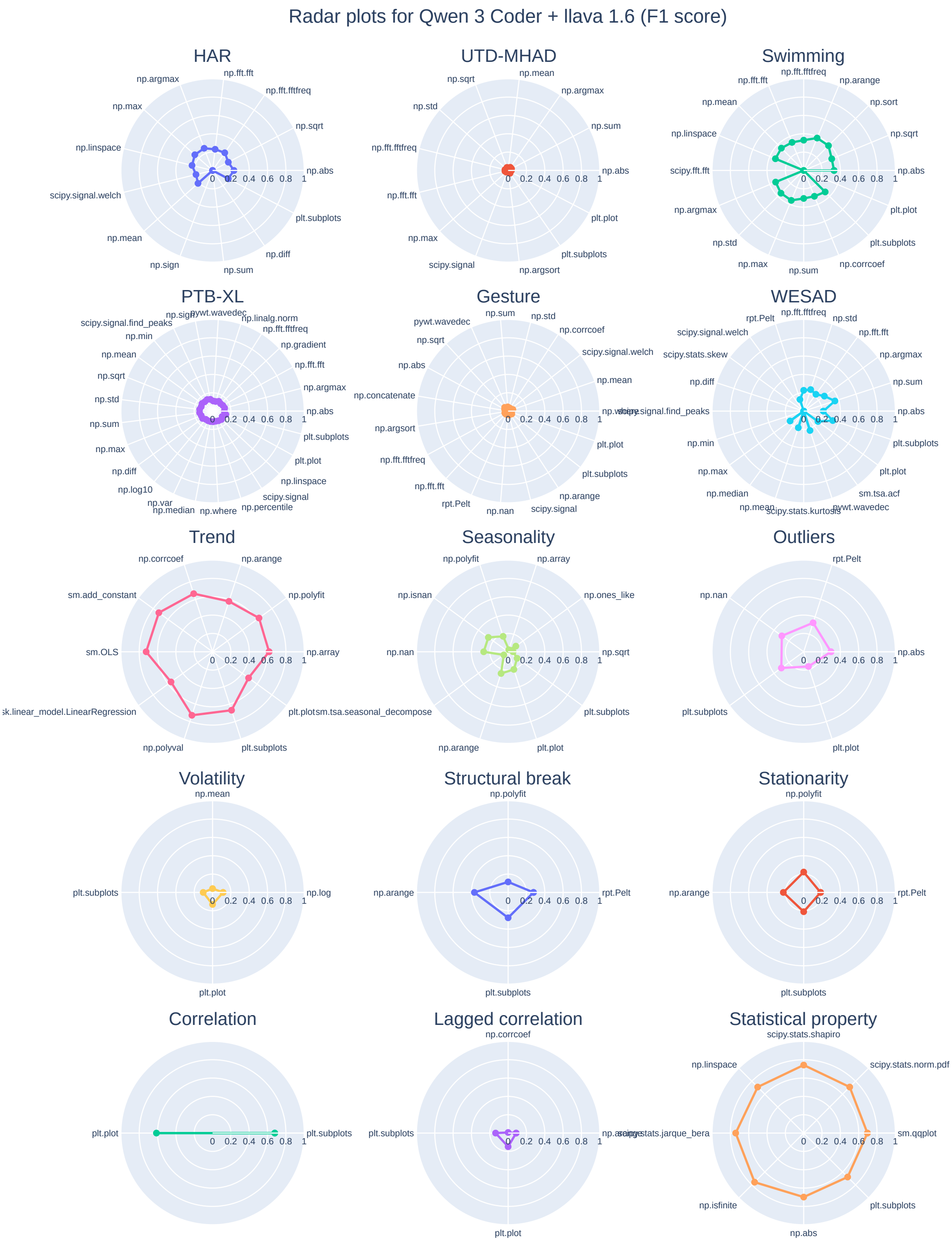


Figure 12: Qwen 3 Coder + llava 1.6

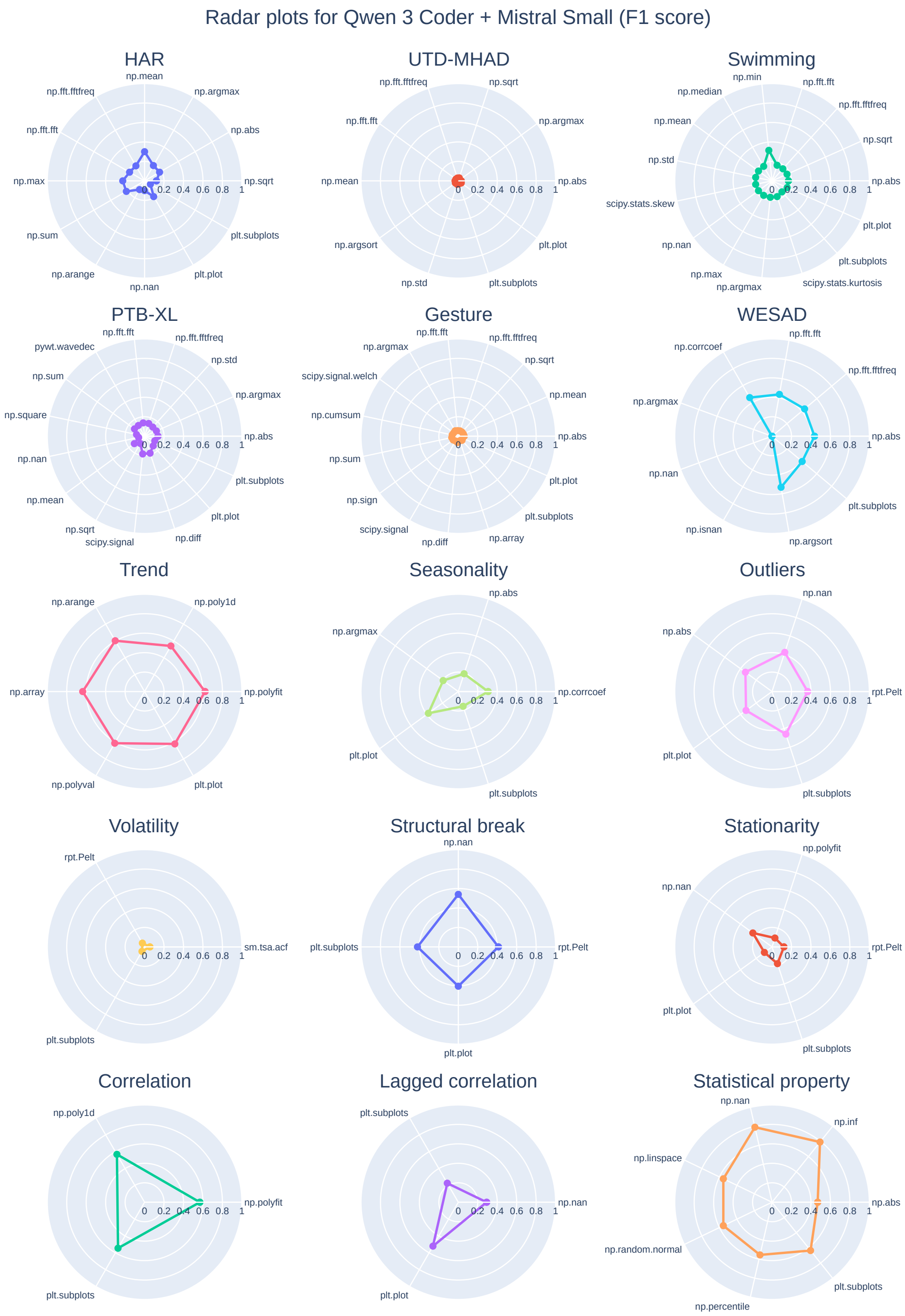


Figure 13: Qwen 3 Coder + Mistral Small

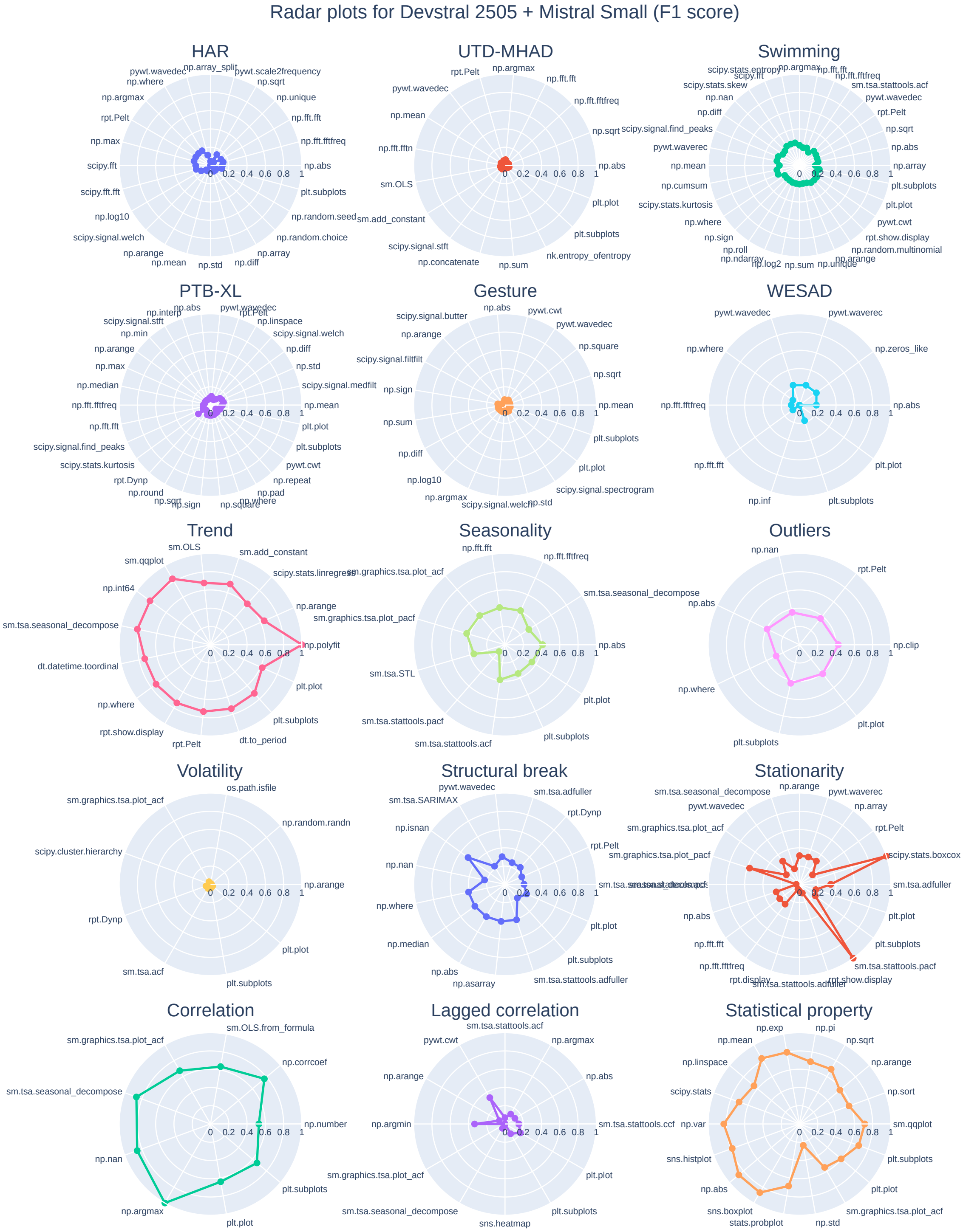


Figure 14: Devstral 2505 + Mistral Small

E WEST Tokamak Examples

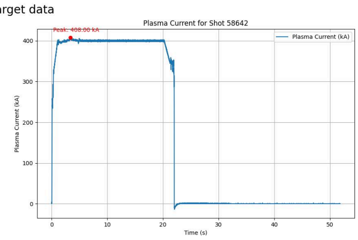
User Query

For shot number 58642 plot the plasma current over the whole discharge and mark the time of its peak value

</> Code Generation

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
result = []  # Initialize plots or pandas dataframes list
df = pd.read_csv("./data.csv", index_col=0)
# Filter data for shot number 58642
shot_data = df.loc[58642]
# Plot plasma current over the whole discharge
plt.figure(figsize=(10, 6))
plt.plot(shot_data.index, shot_data['SMAG_IP'],
 label='Plasma Current (kA)')
plt.title(f'Plasma Current for Shot {58642}')
plt.xlabel('Time (s)')
plt.ylabel('Plasma Current (kA)')
plt.grid(True)
# Find the index of the peak plasma current
peak_index = shot_data['SMAG_IP'].idxmax()
peak_value = shot_data['SMAG_IP'].max()
# Mark the peak value on the plot
plt.scatter(peak_index, peak_value, color='red', zorder=5)
plt.annotate(f'Peak: {peak_value:.2f} kA', (peak_index,
peak_value),
             textcoords="offset points", xytext=(10,10),
             ha='center', color='red')
# Add legend
plt.legend()
result.append(plt)
```

Target data



Answer Generation

WEST Tokamak Assistant ⚡

for shot number 58642 plot the plasma current over the whole discharge and mark the time of its peak value

✓ Graph Outputs

Based on the provided data for shot number 58642, the plasma current over the entire discharge is plotted on the y-axis (in kiloAmperes, kA) against time (in seconds) on the x-axis. The plasma current reaches its peak value of 408.00 kA at approximately 3 seconds. This peak is marked with a red dot on the graph. The plasma current then stabilizes around 400 kA before dropping sharply to near zero at around 23 seconds, where it remains for the rest of the discharge duration.

Figure 15: In this example, a user queries plasma current analysis for shot number 58642. The agent identifies the relevant signal, generates executable code, identifies the peak value (408.00 kA at approximately 3 seconds), and provide analysis of the plasma current evolution.

User Query

is pulse **57287 disrupted** or **not disrupted**

Code Generation

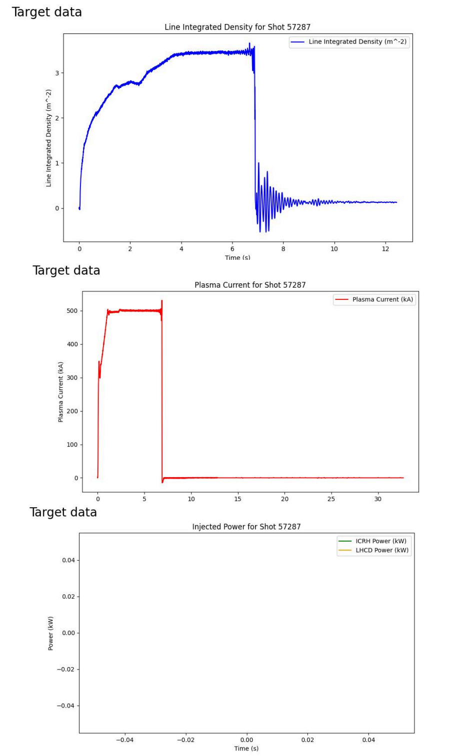
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
result = [] # Initialize plots or pandas dataframes list
df = pd.read_csv("./data.csv", index_col=["shot_id", "time"])
 # Load sensor

# Data processing
shot_id = 57287
shot_data = df.loc[shot_id]

# Plot line integrated density
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(shot_data.index, shot_data['GINTLIDRT%3'],
label='Line Integrated Density (m^-2)', color='blue')
ax.set_xlabel('Time (s)')
ax.set_ylabel('Line Integrated Density (m^-2)')
ax.set_title(f'Line Integrated Density for Shot {shot_id}')
ax.legend()
result.append(fig)

# Plot plasma current
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(shot_data.index, shot_data['SMAG_IP'], label='Plasma
Current (kA)',
 color='red')
ax.set_xlabel('Time (s)')
ax.set_ylabel('Plasma Current (kA)')
ax.set_title(f'Plasma Current for Shot {shot_id}')
ax.legend()
result.append(fig)

# Plot injected power
fig, ax = plt.subplots(figsize=(10, 6))
ax.plot(shot_data.index, shot_data['SICHPTOT'], label='ICRH Power
 (kW)',
color='green')
ax.plot(shot_data.index, shot_data['GPHYB%3'], label='LHCD Power
 (kW)',
color='orange')
ax.set_xlabel('Time (s)')
ax.set_ylabel('Power (kW)')
ax.set_title(f'Injected Power for Shot {shot_id}')
ax.legend()
result.append(fig)
```



Answer Generation

WEST Tokamak Assistant

is pulse 57287 disrupted or not disrupted

Graph Outputs

Based on the provided data, shot 57287 experienced a disruption. Here's why:

1. **Plasma Current (SMAG_IP)**: The plasma current rapidly drops to zero around 6 seconds, which is a clear indicator of a disruption. A stable plasma discharge would maintain a relatively constant current during the flat-top phase.
2. **Line Integrated Density (GINTLIDRT_3)**: The line-integrated density also shows a significant drop to near zero at the same time as the plasma current. This further confirms the disruption, as the plasma density would not abruptly decrease in a non-disrupted shot.
3. **Power Injected by ICRH (SICHPTOT) and LHCD (GPHYB_3)**: The injected power data does not show any significant values, suggesting that heating power was not applied or was turned off before the disruption. However, the lack of power injection could also be a result of the disruption, as the control system might shut down heating power in response to a disruption event.

Given these observations, it is clear that shot 57287 was disrupted around 6 seconds into the discharge.

Figure 16: This example is used to assess the agent ability to classify and analyze plasma stability related questions. The assistant generates plots, analyses multiple diagnostic signals, and provides explanation why this shot was classified as disrupted. While it correctly identifies the shot as disrupted, it also attempts to visualize signals that are not directly relevant to disruption detection.