ClimateAgent: Multi-Agent Orchestration for Complex Climate Data Science Workflows

Anonymous authors
Paper under double-blind review

Abstract

Climate science demands automated workflows to transform comprehensive questions into data-driven statements across massive, heterogeneous datasets. However, generic LLM agents and static scripting pipelines lack climate-specific context and flexibility, thus, perform poorly in practice. We present ClimateAgent, an autonomous *multi-agent* framework that orchestrates end-to-end climate data analytic workflows. ClimateAgent decomposes user questions into executable sub-tasks coordinated by an Orchestrate-Agent and a Plan-Agent; acquires data via specialized Data-Agents that dynamically introspect APIs to synthesize robust download scripts; and completes analysis and reporting with a Coding-Agent that generates Python code, visualizations, and a final report with a built-in self-correction loop. To enable systematic evaluation, we introduce Climate-Agent-Bench-85, a benchmark of 85 real-world tasks spanning atmospheric rivers, drought, extreme precipitation, heat waves, sea surface temperature, and tropical cyclones. On Climate-Agent-Bench-85, ClimateAgent achieves 100% task completion and a report quality score of 8.32, outperforming GitHub-Copilot (6.27) and a GPT-5 baseline (3.26). These results demonstrate that our multi-agent orchestration with dynamic API awareness and self-correcting execution substantially advances reliable, end-to-end automation for climate science analytic tasks.

1 Introduction

The rapid pace of climate change and the growing severity of its impacts have created an urgent need to advance data-centric climate science, which could deliver timely insights for adaptation and policy (Deelman et al., 2009; Overpeck et al., 2011; King et al., 2009). Automating the workflows in climate science can translate comprehensive analytical questions into executable pipelines, enabling rapid, reproducible analysis of complex environmental phenomena and supporting extreme-event forecasting, impact assessment, and adaptation planning. On the other hand, such analytic workflows need to process special climate datasets (e.g., datasets from Copernicus climate data store (CDS) and the European centre for medium-range weather forecasts (ECMWF)) with high volume, heterogeneity, and complexity, where intelligent automation has become essential for processing and integrating these datasets at scale (Hersbach et al., 2020; Benestad et al., 2017; Buizza et al., 2018). In this paper, we aim to explore *how AI agents can perform complex climate-science analytical tasks through carefully designed agentic orchestration.*

Building an AI-driven climate agent is a compelling and crucial effort since the stakes of climate change demand faster and more intelligent ways to derive insights from the complex data. The accelerating pace of global change and the severity of its impacts suggest that climate scientists and policymakers urgently need timely, data-driven information for mitigation and adaptation decisions (Overpeck et al., 2011; King et al., 2009). At the same time, climate science workflows need to process massive and diverse datasets — from multi-model simulations to satellite and in-situ observations — and turning this data deluge into useful information has become a bottleneck for discovery and decision-making. A large language model (LLM) based AI agent can automate complex analytical workflows to address this bottleneck, dramatically accelerating analysis that would otherwise take a team of experts days or weeks. By translating high-level scientific questions into executable pipelines, such an agentic paradigm could enable rapid, reproducible analysis of

complicated climate phenomena like extreme events, climate impacts, and future scenarios (Deelman et al., 2009; Overpeck et al., 2011). In essence, an AI climate agent can serve as a productive research assistant that integrates data from various sources on-the-fly and explores many potential hypotheses iteratively, which would augment human scientists' abilities, i.e., allowing them to focus on interpretation and strategy, and accelerate the cycle of the analytic workflow (King et al., 2009; Reichstein et al., 2019). By empowering more interactive and comprehensive data exploration, such an agent could usher in a new paradigm for climate science, where insights emerge at the pace of computational power rather than human labor.

On the other hand, developing a robust climate agent is challenging due to several inherent complexities of this domain, where data volume and heterogeneity are primary obstacles: modern climate datasets are enormous in size and varied in format. For example, a single state-of-the-art reanalysis (ERA5) encompasses petabytes of multidimensional data (Hersbach et al., 2020), and observational records, climate model outputs, and remote sensing products each come with different resolutions, units, and conventions (Benestad et al., 2017). A simple, hand-crafted pipeline or off-the-shelf use of LLMs (e.g., standard zero-shot LLM approaches) will quickly break down when faced with such diversity and complexity. Note that many climate workflows require on-the-fly decision making and expert knowledge — for instance, selecting appropriate data sources, applying bias corrections, or choosing relevant statistical tests to determine significance. A rigid, hand-crafted solution cannot easily accommodate these nuanced choices — attempts to force flexibility into static pipelines often lead to brittle, error-prone processes, highlighting the need for more intelligent, adaptive automation (Buizza et al., 2018).

In this paper, we view automating complex climate scientific workflows as a specialized form of planning and data-processing code generation. Recent LLM-based agentic systems show promising performance, but those applied to scientific domains still face critical shortcomings (Austin et al., 2021). Most current agents use generic, domain-agnostic LLMs that miss the specialized requirements of climate science (Wang et al., 2024; Yao et al., 2022). Concretely, a general LLM agent could be unaware of climate data access APIs, data formats, and valid parameter choices, while static scripts and libraries lack the flexibility to handle new scientific questions or evolving data sources. These limitations result in high error rates for LLM-generated code and a steep learning curve for non-expert users. More importantly, generic approaches cannot support the iterative, hypothesis-driven nature of climate study — they fail to autonomously handle multi-step inquiries where initial results could spur new sub-questions — leaving a substantial gap in achieving truly end-to-end, autonomous climate analysis without requiring human intervention.

To address these gaps, we introduce ClimateAgent, an autonomous multi-agent framework specially designed for climate science workflows. Our approach employs a multi-agent orchestration strategy: a comprehensive climate query is first decomposed into a structured sequence of sub-tasks, where each sub-task is executed by a specialized agent. This modular design injects domain-specific knowledge at every step and dynamically adapts as the workflow progresses. Additionally, ClimateAgent is also equipped with robust error handling and self-correction, enabling it to detect failures, recover, and adjust plans without human intervention in the loop. Concretely, we make the following contributions:

Contribution 1. We propose a multi-agent orchestration paradigm to support complex climate data science workflows, which comprises the following key agents:

- **Agents for planning and orchestration**: We introduce an Orchestrate-Agent for workflow management and a Plan-Agent for task decomposition. Concretely, the Orchestrate-Agent manages experiment directories and persistent context, while the Plan-Agent interprets the user request, formulates a detailed execution plan, breaks down the high-level goal into discrete subtasks, and delegates each to the appropriate specialist agent. Together, these two agents provide top-level oversight, adjusting the plan as needed and ensuring the overall workflow stays on track, including handling runtime issues or re-planning when necessary.

- **Agents for climate data acquisition**: We implement a set of Data-Agents each tailored to a specific data source and handles data retrieval by dynamically introspecting its target API — e.g., fetching the latest valid parameters or dataset metadata at runtime — and then generating robust download scripts. This capability allows users to adapt to evolving datasets and API utilization, preventing errors such as invalid parameter use or format mismatches during data acquisition.

- **Agents for programming and visualization**: We include a set of Coding-Agents that generate Python code for data analysis and generate final reports, complete with text summaries and visualizations. The Coding-Agents implement a self-correction loop to debug code based on execution feedback.

Together, these agents form a cohesive system that can autonomously manage the entire climate data science life-cycle from data acquisition to final analysis, making sophisticated climate science accessible and efficient.

Contribution 2. To systematically evaluate the performance over the climate data science tasks, we introduce Climate-Agent-Bench-85, a benchmark of 85 real-world climate workflow tasks spanning six domains, including atmospheric rivers (AR), drought (DR), extreme precipitation (EP), heat waves (HW), sea surface temperature (SST), and tropical cyclones (TC). Each task is specified in natural language with an explicit scientific objective, required datasets, mandatory external tools (e.g., `TempestExtremes`), and strict output contracts (filenames and formats), driving multi-step pipelines from data acquisition through processing, analysis, and visualization. For reproducibility, every task includes a curated reference solution with a validated Python code base and a human-readable report with figures, and tasks are stratified by difficulty — *easy* (single-source), *medium* (multi-source), and *hard* (external-tool integration with dynamic parameters) — to support controlled comparisons across complexity levels.

Contribution 3. We comprehensively evaluate the proposed multi-agent framework, ClimateAgent, on Climate-Agent-Bench-85, where the experimental results indicate that ClimateAgent demonstrates the ability to autonomously plan and execute the entire workflow. To be specific, our system achieves a 100% success rate in generating the report across the benchmark, with an overall report quality score of 8.32, compared to 6.27 for GitHub-Copilot and 3.26 for the GPT-5 baseline. We believe ClimateAgent significantly reduces the manual effort and specialized expertise required, demonstrating a powerful new paradigm for AI-driven climate data science discovery and advancing the state-of-the-art in automated climate research.

2 Related Work

LLM-Based Agentic Systems and Scientific Automation. Large language models such as GPT-3 (Brown et al., 2020), Llama 2 (Touvron et al., 2023), and PaLM (Chowdhery et al., 2023) have become the backbone for agentic systems that interpret user instructions, plan tasks, and interact with external APIs (Yao et al., 2022; Shinn et al., 2023). Frameworks like AutoGPT (Richards, 2023), LangChain (Chase, 2023), and CrewAI (CrewAI, 2023) enable multi-step workflows and collaborative agents, while recent advances have extended these systems with memory, tool usage, and inter-agent collaboration capabilities (Chen et al., 2024; Park et al., 2023). In scientific domains, systems such as ChemCrow (Bran et al., 2024) and SciFact (Wadden et al., 2020) demonstrate automated protocol generation and fact checking. However, these approaches are generally evaluated in synthetic domains or require extensive human oversight. What remains missing is a system that combines domain-specific knowledge integration, persistent workflow state management, and robust error recovery for end-to-end scientific analysis with live, evolving data sources.

Climate Science Workflow Automation. Workflow automation in climate science has evolved from generic orchestrators like Kepler (Altintas et al., 2004) and Apache Airflow to domain-adapted engines such as Cylc (Oliver et al., 2018) and ESMValTool (Righi et al., 2020). Programmatic access to climate datasets is enabled by libraries like cdsapi, ecmwf-api-client (Copernicus Climate Change Service (C3S), 2019; European Centre for Medium-Range Weather Forecasts (ECMWF), 2022), and processing interfaces like CDO and xarray (Hoyer & Hamman, 2017). Recent specialized systems have begun targeting autonomous climate analysis: ClimSim-Online (Sridhar et al., 2023) explored goal-driven automation for climate impact modeling, while TorchClim (Fuchs et al., 2024) and EarthML (Project, 2020) focus on deep learning-powered workflows. However, existing systems either require manual intervention at each workflow stage or lack the robustness to handle the dynamic nature of climate APIs and heterogeneous data sources, preventing truly autonomous end-to-end analysis.

Autonomous Research and Multi-Agent Planning. Advanced LLM architectures incorporating meta-prompting (Zhang et al., 2025c), chain-of-thought reasoning (Wei et al., 2022), and self-correction feedback loops (Kamoi et al., 2024) have improved task decomposition and reliability in autonomous research sys-

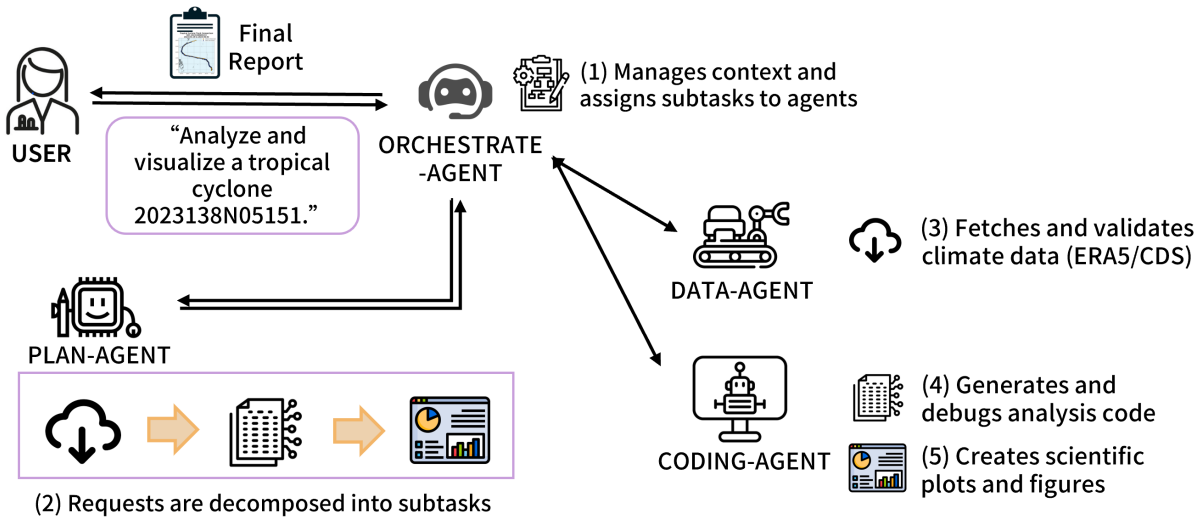


Figure 1: Overview of the ClimateAgent system architecture. The workflow illustrates how user queries are decomposed and processed by specialized agents, with robust error recovery and context management, to produce comprehensive climate science reports.

tems. Prompt-based approaches like PromptCast (Xue & Salim, 2022) and LLMTime (Gruver et al., 2023) demonstrate zero-shot capabilities in time series analysis, while automated debugging agents (Gao et al., 2024) enhance code reliability. However, the gap between high-level planning capabilities and execution-level robustness remains unaddressed: few systems successfully translate broad user goals into complex multi-agent plans with error-recoverable execution for scientific APIs, particularly in climate science.

To address these gaps, i.e., domain-agnostic agents, brittle workflow execution, and lack of comprehensive evaluation benchmarks, we introduce ClimateAgent, a specialized multi-agent framework that embeds climate expertise at every workflow stage while maintaining the flexibility to recover from inevitable failures.

3 ClimateAgent

Climate science workflows present a fundamental challenge for automation: researchers must coordinate multiple interdependent tasks across heterogeneous data sources, each requiring specialized domain knowledge. A typical workflow begins with identifying and acquiring data through diverse APIs — reanalysis products, forecast models, observational datasets — then proceeds through multi-dimensional data processing, statistical analysis, and finally report generation. Each phase demands expertise in domain-specific conventions: climate data APIs impose unique constraints on spatial and temporal queries, scientific computing libraries require precise parameter configurations, and analysis must respect the statistical properties of geophysical data. Existing LLM-based approaches cannot handle this complexity: single-agent systems lack the specialized knowledge required at each stage, while monolithic code generation cannot recover from API violations, parameter errors, or dependency failures.

We observe that the structure of climate workflows naturally suggests a solution. Rather than attempting monolithic generation, we can decompose complex analyses into specialized subtasks — mirroring how climate researchers organize their own work into data acquisition, processing, analysis, and reporting phases. By assigning each subtask to an expert agent with targeted domain knowledge, and coordinating execution through shared context that accumulates artifacts and enables iterative refinement, we leverage LLMs' code generation capabilities while building in the error recovery mechanisms that long scientific workflows require.

We present ClimateAgent, an autonomous multi-agent framework that executes end-to-end climate analyses without human intervention. ClimateAgent decomposes workflows into specialized agents that collaborate through shared context and adaptive feedback loops. The Orchestrate-Agent coordinates execution

while the PLAN-AGENT decomposes queries into executable subtasks. DATA-AGENTS introspect climate APIs to generate validated download scripts, and CODING-AGENTS produce analysis code with built-in debugging and self-correction. To achieve truly autonomous scientific workflows, CLIMATEAGENT realizes three core capabilities that address the fundamental requirements of climate analysis automation:

- **Coordinated Task Planning.** The PLAN-AGENT decomposes comprehensive climate queries into structured subtasks and delegates each to specialized agents with targeted domain expertise, forming a cooperative division of labor that improves the completeness and quality of the resulting analysis and reports.
- **Contextual Coordination.** Agents operate on a shared, persistent context that enables cross-step communication and propagation of intermediate results, dynamically adapting as the workflow progresses and maintaining methodological consistency across multi-stage analyses.
- **Adaptive Self-Correction.** CLIMATEAGENT incorporates built-in error detection and recovery mechanisms that allow agents to diagnose failures, revise plans, and adapt without human intervention, ensuring robustness under evolving data and API conditions.

In this section, we first formalize climate report generation as sequential state transformation (§3.1), then describe how our three-layer architecture achieves **Coordinated Task Planning** (§3.2), how persistent context enables **Contextual Coordination** (§3.3), and how complementary error recovery strategies provide **Adaptive Self-Correction** (§3.4). Finally, we synthesize these mechanisms into a unified orchestration algorithm (§3.5).

3.1 Problem Formulation

We formalize climate workflow execution as a sequential state transformation. Given a task T, the system produces a scientific report R as the final output of a multi-stage analytical workflow through coordinated multi-agent operations that maintain a persistent workflow context $\mathcal{C}_i$ accumulating all generated artifacts:

$$\mathcal{C}_i = \{\text{task} : T, \text{plan} : P, \text{code} : \{c_j\}_{j=1}^{i}, \text{data} : \{d_j\}_{j=1}^{i}, \text{results} : \{r_j\}_{j=1}^{i}, \text{logs} : \{l_j\}_{j=1}^{i}\}$$

The PLAN-AGENT decomposes T into an ordered sequence of subtasks $P = [s_1, s_2, \ldots, s_n]$, each specifying the action, required data, and target agent. Specialized agents A_k execute subtasks via the transition:

$$\mathcal{C}_i = \text{Execute}(\mathcal{C}_{i-1}, s_i, A_k)$$

This design achieves *workflow coherence through context accumulation*: each agent operates on the complete history of prior decisions and artifacts. The context serves as (1) an inter-agent communication protocol, (2) a checkpoint for workflow resumption, and (3) a provenance record for reproducibility.

3.2 Multi-Agents for Coordinated Task Planning

To realize the coordinated task planning capability, CLIMATEAGENT employs a three-layer hierarchical architecture (see Figure 1) that mirrors how climate scientists organize collaborative investigations. Specialized agents at each layer encode distinct domain expertise: the PLAN-AGENT and ORCHESTRATE-AGENT coordinate workflow execution, DATA-AGENTS handle data acquisition, and CODING-AGENTS perform analysis and synthesis.

Agents for Planning and Orchestration. ORCHESTRATE-AGENT manages workflow execution (Step 1 in Figure 1) by creating timestamped experiment directories, persisting context, and coordinating agent invocation. PLAN-AGENT decomposes queries into executable subtasks using climate-domain reasoning: it recognizes standard analysis patterns (climatology → anomalies → extremes → report), dataset dependencies (reanalysis vs. forecasts), and temporal constraints (initialization dates, lead times, aggregation windows).

Agents for Climate Data Acquisition. DATA-AGENTS interface with heterogeneous climate data sources (Step 2) through specialized implementations. The CDS variant uses `cdsapi` to access the Copernicus Climate Data Store, validating variables, time ranges, and spatial domains. The ECMWF variant uses

`ecmwf-api-client` to retrieve S2S forecasts, encoding knowledge of model origins (ECMWF, NCEP, JMA), parameter types (pressure-level, surface, daily-averaged), and forecast conventions.

Agents for Programming and Visualization. Coding-Agents generate Python code (Steps 3-5) for data processing and report visualization. They incorporate expertise in climate libraries (xarray, cartopy, cf-python), domain-appropriate statistical methods (anomaly calculations, running means, extreme value statistics), and scientific computing practices (vectorized operations, memory management, unit handling).

This three-layer architecture separates concerns that require distinct expertise: data acquisition demands API knowledge, processing requires computational skills, and reporting requires synthesis ability. Encoding domain expertise at each layer achieves modularity and reliability.

3.3 Contextual Coordination

Building upon the hierarchical architecture in §3.2, ClimateAgent achieves the second core capability, i.e., contextual coordination, by maintaining a persistent and interpretable workflow memory shared across all agents. This shared context records plans, data artifacts, and execution outputs, enabling continuity, communication, and reproducibility throughout the workflow. Formally, the persistent workflow context $\mathcal{C}_i$ serves as the central mechanism that connects agents, supports error recovery, and preserves the system state. Context functions as both a communication protocol and a state management system.

Context Evolution. In this procedure, each agent receives $\mathcal{C}_{i-1}$ and produces $\mathcal{C}_i$ by appending new artifacts (scripts, outputs). This monotonic accumulation preserves all information across agent transitions. For example, Coding-Agent accesses original data files, processed results, and prior code when generating visualizations.

Context-Driven Code Generation. The agents should then leverage context to generate coherent code. Concretely, Coding-Agent examines $\mathcal{C}_{i-1}$ to discover available files, understand data structure, and maintain consistent naming conventions. This prevents common errors: hard-coded paths, incorrect variable assumptions, and redundant computations.

Serialization and Persistence. We serialize context to JSON after each subtask, enabling: (1) *workflow resumption* from the last completed subtask after interruption; (2) *reproducibility* by preserving complete decision history; (3) *debugging* without re-executing earlier stages. This persistent state transforms independent agents into a coherent, traceable workflow engine.

3.4 Adaptive Self-Correction

Building on the contextual coordination mechanisms in §3.3, ClimateAgent realizes the third core capability, i.e., adaptive self-correction, which provides robustness and adaptability under real-world scientific constraints. Note that climate workflows could face unexpected execution failures: APIs impose inconsistent parameter rules, data availability fluctuates, and generated code contains subtle errors in coordinate transformations, unit conversions, or array indexing. Single mistakes cascade through entire workflows in baseline LLM systems.

ClimateAgent employs three complementary error recovery strategies. *Multi-candidate generation* addresses API variability: Data-Agent generates $m = 8$ candidate download scripts with varying interpretations (spatial bounds, temporal aggregations, variable selections) and executes them sequentially until one succeeds. *Iterative refinement* handles runtime errors: Coding-Agent retries up to $R_{\max} = 3$ times, incorporating diagnostics from previous failures, with up to 5 debug iterations per candidate. *LLM-based semantic validation* catches subtle correctness issues (wrong statistical tests, incorrect climatological periods) that produce scientifically invalid results without runtime errors.

These strategies complement each other: multi-candidate generation explores hard-to-predict parameter spaces, iterative refinement fixes implementation bugs using error feedback, and semantic validation ensures scientific correctness. Together, they enable reliable execution of complex workflows involving unpredictable APIs and intricate computations.

Algorithm 1: ClimateAgent Orchestration Workflow

```
Input: Climate research task T
Output: Scientific report R
Initialize experiment directory and context C_0;
// Task decomposition phase
P ← PLAN-AGENT.decompose(T)
C_0.plan ← P
// Sequential subtask execution with error recovery
for i ← 1 to |P| do
    // Current subtask
    s_i ← P[i]
    // Route to appropriate agent
    A_k ← select_agent(s_i.type)
    retry_count ← 0
    success ← false
    // Retry loop with maximum attempts
    while retry_count < R_max and not success do
        if A_k = DATA-AGENT then
            // Strategy 1: Multi-candidate generation for downloads
            S ← A_k.generate_candidates(s_i, C_{i-1}, m = 8)
            foreach c ∈ S do
                result ← execute(c, exp_dir)
                if result.success then
                    C_i ← update_context(C_{i-1}, s_i, c, result)
                    success ← true
                    break
        else
            // Strategy 2 & 3: Iterative refinement + semantic validation
            c ← A_k.generate(s_i, C_{i-1}, error_history)
            // Semantic validation
            is_valid ← LLM_validate(c, s_i, T)
            // Runtime execution
            result ← execute(c, exp_dir)
            if result.success then
                C_i ← update_context(C_{i-1}, s_i, c, result)
                success ← true
            else
                error_history.append(result.error)
                retry_count ← retry_count + 1
// Extract final report from coding agent output
R ← C_n.results[final_report]
return R
```

3.5 Orchestration Algorithm

Bringing together the capabilities of Coordinated Task Planning, Contextual Coordination, and Adaptive Self-Correction, Algorithm 1 synthesizes the three core capabilities into a unified orchestration process (Steps 1-5 in Figure 1). This orchestration approach ensures that complex climate science workflows are executed reliably through: (1) *context-driven coordination* enabling agents to build on prior work, (2) *multi-strategy error recovery* handling diverse failure modes, (3) *complete state persistence* supporting reproducibility and debugging, and (4) *graceful degradation* with clear failure reporting when tasks cannot be completed. These mechanisms collectively enable researchers to focus on scientific questions rather than technical implementation details.

4 Climate Agentic Workflow Benchmark

Another critical question for applying LLM agents for climate workflows is: *how can one systematically evaluate whether these design choices translate into reliable, high-quality scientific outputs?* This requires a benchmark that captures the authentic complexity of climate research — diverse phenomena, heterogeneous data sources, multi-step reasoning, and domain-specific correctness criteria.

Evaluating LLM-based systems for scientific workflows requires benchmarks reflecting authentic research complexity. Climate workflows demand domain-specific API knowledge, multi-step reasoning across data acquisition and analysis, integration with specialized tools, and scientifically interpretable outputs. Existing benchmarks like DataSciBench (Zhang et al., 2025a) and MASSW (Zhang et al., 2025b) focus on structured data science tasks but lack the domain complexity and real-world API integration of climate research.

To address this gap, we introduce Climate-Agent-Bench-85, comprising 85 end-to-end workflow tasks across six climate phenomena. Each task requires executable code that retrieves data from operational APIs, processes multi-dimensional datasets, performs domain-appropriate analyses, and generates publication-quality reports. The benchmark evaluates code generation, workflow planning, error recovery under API constraints, and scientific correctness.

This benchmark operationalizes the three capability dimensions defined in §3 — planning, persistent context, and robustness — by translating them into measurable workflow tasks. The following sections describe how these capabilities are reflected in task design and evaluated through a unified protocol.

4.1 Design Principles and Capability Mapping

To empirically evaluate the three system capabilities introduced in §3, Climate-Agent-Bench-85 is constructed around three guiding principles that mirror the design goals of ClimateAgent:

- **Coordinated Task Planning.** Tasks require multi-stage decomposition — from data retrieval to analysis and visualization — so that effective workflow planning can be distinguished from ad-hoc single-step reasoning.
- **Contextual Coordination.** Many tasks contain cross-step dependencies, such as derived variables, climatological baselines, or intermediate files that must be reused downstream, testing whether an agent can maintain and propagate state across iterative reasoning and execution.
- **Adaptive Self-Correction.** Tasks involve real-world API and tool interactions (e.g., ECMWF API, TempestExtremes) under realistic parameter constraints, stressing the system's ability to detect, recover, and adapt to execution or formatting errors.

4.2 Task Domains and Scientific Coverage

Climate-Agent-Bench-85 spans six climate phenomena representing diverse analysis patterns:

- **Atmospheric Rivers (AR, 15 tasks)** require computing integrated vapor transport (IVT) from multi-level wind and humidity fields, applying physical thresholds, identifying spatial regions, and tracking temporal evolution. These tests involve vertical integration, spatial pattern recognition, and trajectory analysis with wraparound coordinates.
- **Drought (DR, 15 tasks)** compute multi-timescale indices (SPI, soil moisture anomalies) requiring 30-year climatological baselines, statistical standardization handling seasonal cycles, and categorical severity visualization. These evaluate temporal aggregation, statistical edge cases (zero variance, missing data), and standard visualization conventions.
- **Extreme Precipitation (EP, 15 tasks)** analyze precipitation extremes through percentile metrics, spatial extent, and multi-day event evolution. Workflows aggregate sub-daily data, identify threshold exceedance (25–250 mm/day), and visualize progression. These assess temporal conversions, unit management, and multi-panel figures.
- **Heat Waves (HW, 10 tasks)** identify prolonged high-temperature events using multiple frameworks (absolute/percentile thresholds, duration criteria, wet-bulb temperature). These tests include multi-criteria detection, boolean logic, binary mask handling, and custom colormaps.

- **Sea Surface Temperature (SST, 15 tasks)** examine patterns, anomalies, and trends, including ENSO indices and marine heat waves. Workflows compute climatological anomalies, identify warm/cold events, and analyze spatial-temporal evolution. These require ocean-atmosphere process understanding, regional indices (Niño 3.4), and diverging color schemes.
- **Tropical Cyclones (TC, 15 tasks)** use TempestExtremes (Ullrich et al., 2021) with ERA5 and IBTrACS data for detection, tracking, and intensity analysis. These require dynamic parameter configuration, command-line tool integration, trajectory stitching, and multi-source validation — testing tool I/O, subprocess management, and error recovery.

These domains cover atmospheric (AR, TC), hydrological (DR, EP), land surface (HW), and oceanic (SST) processes, spanning diverse scales and methodologies. AR and TC tasks test planning through multi-stage pipelines; DR and SST test context through climatological dependencies; EP and HW test robustness through complex thresholding and conversions.

4.3 Task Construction Methodology

Expert-Driven Design. Three atmospheric science graduate students from our institution (co-authors of this work) with combined expertise spanning synoptic meteorology, climate dynamics, and computational climate science systematically designed all tasks. The design process followed an iterative methodology:

- **Domain Selection**: Experts identified six phenomena that (a) represent diverse spatial and temporal scales (from daily precipitation extremes to seasonal ENSO patterns), (b) require different data sources and processing pipelines, (c) reflect common research workflows in operational and academic climate science, and (d) present varying computational and algorithmic challenges.
- **Task Diversification**: Within each domain, we designed tasks to maximize coverage of analysis patterns, data sources (ERA5, S2S forecasts, OISST, IBTrACS), spatial domains (regional to global), and temporal scales (daily to monthly). Tasks explicitly avoid redundancy — each presents unique requirements in data handling, statistical methods, or visualization conventions.
- **Real-World Grounding**: All tasks are based on actual research workflows employed in published climate studies or operational forecasting. For example, AR detection follows algorithms from Pan & Lu (2019), drought analysis implements WMO-standardized SPI methodology, and TC tracking uses established TempestExtremes configurations. This grounding ensures tasks reflect authentic scientific practice rather than artificial benchmarks.
- **Specification Refinement**: Initial task descriptions underwent multiple revision cycles to eliminate ambiguity while preserving implementation flexibility. All three experts reviewed each specification to ensure clarity, scientific accuracy, and feasibility.

This expert-driven curation ensures that each task not only reflects real-world research practice, but also targets specific capability dimensions introduced in §3. The following subsection further stratifies these tasks according to their expected planning depth, context dependence, and robustness requirements.

4.4 Task Complexity Stratification

We stratify tasks by workflow complexity determined by subtask steps, data sources, external tools, and algorithmic sophistication:

- **Easy Tasks (n=25, 30%)** require single-source acquisition with straightforward processing — one API call (ERA5/OISST), standard xarray operations, basic statistics, single-panel visualization. These tasks test fundamental capabilities: API parameter construction, coordinate handling, unit conversions, and basic plotting.
- **Medium Tasks (n=30, 35%)** involve multi-source data integration or multi-step workflows — coordinating multiple API calls, handling data heterogeneity (mismatched grids, resolutions, conventions), multi-stage pipelines (compute derived variable → identify events → track evolution), multi-panel figures. Medium tasks evaluate the system's ability to maintain workflow coherence across dependent steps, manage intermediate outputs, and ensure consistency.

- **Hard Tasks (n=30, 35%)** require external tool integration with dynamic parameterization — using TempestExtremes or CDO where parameters depend on runtime-discovered data characteristics, managing tool I/O, coordinating heterogeneous sources (forecasts + observations + tool outputs), implementing sophisticated algorithms (trajectory stitching, multi-criteria detection). These tasks test tool integration, subprocess management, and error recovery.

This distribution reflects realistic research workloads and aligns with capability dimensions: *easy* tasks assess planning, *medium* tasks require context maintenance, *hard* tasks challenge robustness.

4.5 Evaluation Protocol

To assess end-to-end system performance on Climate-Agent-Bench-85, we design a multi-dimensional evaluation protocol aligned with the scientific communication standards of the climate community. This framework centers on a unified *report score*, which quantifies overall output quality on a 1–10 scale across four dimensions critical for scientific communication:

- **Readability**: Clarity, logical flow, accurate use of scientific terminology, and accessibility to the target audience.
- **Scientific Rigor**: Adherence to methodological standards, appropriate statistical analyses, uncertainty quantification, and validity of result interpretation.
- **Completeness**: Coverage of all task requirements, inclusion of relevant contextual information, and delivery of actionable insights.
- **Visual Quality**: Relevance and clarity of figures, appropriate visualization methods, accurate labeling, and professional presentation.

Following established practices in recent evaluation literature (Hada et al., 2024; Li et al., 2025; Zheng et al., 2023), we adopt an LLM-based judging framework for report quality assessment. This approach has demonstrated high correlation with expert human evaluation while enabling scalable and consistent scoring across large benchmarks. Our evaluator employs GPT-4o's multimodal capabilities to assess both textual content and embedded visualizations, comparing system outputs against expert-generated references. This evaluation protocol provides the foundation for the quantitative and qualitative analyses presented in §5.

5 Experiments

Building upon the system capabilities defined in §3 and the benchmark and evaluation framework established in §4, we now empirically examine how well ClimateAgent fulfills its design objectives. Our experiments aim to answer the following three core questions:

- **Q1. Coordinated Task Planning:** Does collaborative division of workflow among specialized agents lead to higher-quality scientific reports and more complete end-to-end workflows compared with standard zero-shot LLM reasoning?
- **Q2. Contextual Coordination:** Can the system maintain cross-step dependencies and methodological consistency throughout multi-stage scientific analyses, ensuring coherent reasoning from data acquisition to interpretation?
- **Q3. Adaptive Self-Correction:** Can our error detection and recovery mechanisms enable the system to autonomously identify failures, revise its plans, and continue execution without human intervention?

To evaluate these hypotheses, we compare ClimateAgent with strong baseline models on the Climate-Agent-Bench-85 benchmark (§4), conducting quantitative, qualitative, and ablation-based analyses for each capability dimension. We first describe our experimental protocol and baseline systems (§5.1), then present quantitative results on task planning (§5.2), qualitative analysis on context coordination (§5.3), and ablation studies demonstrating adaptive self-correction (§5.4).

Table 1: End-to-end report quality (Report Score) by domain.

Domain	GPT-5	Copilot	ClimateAgent
Atmospheric River (AR)	3.05	6.78	**7.32**
Drought (DR)	7.87	6.87	**8.57**
Extreme Precipitation (EP)	0.62	5.58	**8.43**
Heatwave (HW)	3.98	8.30	**9.15**
Sea Surface Temperature (SST)	4.28	8.10	**8.88**
Tropical Cyclone (TC)	0.00	2.65	**7.85**
All Tasks	3.26	6.27	**8.32**

Table 2: Overall Performance Summary on Climate-Agent-Bench-85. All scores are averaged across all tasks on a 1-10 scale.

System	Readability	Scientific Rigor	Completeness	Visual Quality	Report Quality
ClimateAgent	**8.40**	**8.72**	**7.75**	**8.41**	**8.32**
Baseline (GPT-5)	3.48	3.41	2.8	3.34	3.26
Baseline (Copilot)	6.68	6.89	5.62	5.87	6.27

5.1 Experimental Setup

We evaluate ClimateAgent on the Climate-Agent-Bench-85 benchmark using the evaluation protocol defined in §4.5. We evaluate ClimateAgent against two strong baselines:

- **GPT-5 Baseline**: A sophisticated baseline leveraging GPT-5's intrinsic reasoning capabilities with execution validation. This system employs best-of-N sampling (N=4) to generate multiple code candidates per task, executes them in a sandboxed environment, and selects the first successful solution. While capable of multi-step reasoning within single generations, this baseline lacks structured workflow decomposition, domain-specific knowledge integration, and iterative refinement mechanisms.
- **GitHub Copilot Agent Mode**: An agentic baseline utilizing GitHub Copilot's conversational capabilities for multi-turn task execution. This system leverages Copilot's advanced code generation expertise (powered by OpenAI Codex (Chen et al., 2021)) combined with its ability to maintain conversational context, execute code iteratively, and provide refinements based on execution feedback. While representing state-of-the-art general-purpose coding assistance, Copilot relies on broad programming knowledge without the domain-specific climate science expertise, structured workflow decomposition, or specialized agent coordination that characterizes our approach.

Both baseline systems utilize GPT-5 as the foundational LLM, ensuring our comparison isolates the effects of specialized agent coordination versus advanced single-model reasoning with execution validation.

5.2 Experimental Results about Task Planning

To answer Q1, we assess how coordinated task planning impacts end-to-end report generation on Climate-Agent-Bench-85 using the setup in §4 and §5.1. For each of the six climate domains, systems must execute the full data-to-report workflow under the evaluation protocol of §4.5. We compare ClimateAgent against the GPT-5 and Copilot baselines, which share the same underlying LLM but lacks explicit multi-agent decomposition and domain-specialized roles. Results are shown in Tables 1 and 2.

Under the evaluation protocol in §4.5, higher scores on Readability, Scientific Rigor, Completeness, Visual Quality, and overall Report Quality can all be interpreted as downstream consequences of more effective task planning: more coherent decomposition and delegation should yield clearer narratives (Readability), more

appropriate methodological choices (Scientific Rigor), fuller coverage of required steps (Completeness), and better targeted figures (Visual Quality).

Experiment Results and Discussions. We summarize the main results and discussion below:

- **Overall Impact of Coordinated Planning.** Across all 85 tasks, ClimateAgent achieves the highest average Report Quality (**8.32** vs. 6.27 for Copilot and 3.26 for GPT-5; Table 2), indicating that explicit division of labor among specialized agents leads to substantially better end-to-end reports than single-model reasoning with execution validation. These gains are consistent across all six domains (Table 1), with particularly large margins in complex, multi-stage settings such as Extreme Precipitation (8.43 vs. 5.58 vs. 0.62) and Tropical Cyclones (7.85 vs. 2.65 vs. 0.00), where baselines frequently fail to complete multi-step analyses.
- **Completeness and Scientific Rigor as Planning Outcomes.** Beyond overall Report Quality, ClimateAgent also substantially outperforms baselines on **Completeness** (**7.75** vs. 5.62 vs. 2.80) and **Scientific Rigor** (**8.72** vs. 6.89 vs. 3.41). Interpreted through the lens of Q1, higher Completeness reflects that the Plan-Agent decomposes each task into concrete subgoals and assigns them to specialized Data-Agents and Coding-Agents, reducing missing figures, truncated analyses, and omitted discussion that commonly occur in the GPT-5 and Copilot baselines. Higher Scientific Rigor indicates that coordinated planning encourages a more disciplined methodological choices — selecting appropriate datasets, applying physically meaningful aggregations, and justifying parameter choices in a way that aligns with expert expectations — rather than the ad-hoc, inconsistent methods often observed in single-model workflows.
- **Readability and Visual Quality as Planning Effects.** Coordinated planning also improves how results are communicated. As shown in Table 2, ClimateAgent attains higher **Readability** (8.40 vs. 6.68 vs. 3.48) and **Visual Quality** (8.41 vs. 5.87 vs. 3.34) than both baselines. From the perspective of Q1, these gains arise because the Plan-Agent explicitly anticipates the narrative and visual requirements of each task — specifying which diagnostics, figures, and explanatory sections are needed — and delegates them to specialized agents. This leads to reports with clearer structure, better-matched figures, and consistent labeling that directly support the planned analytical storyline. In contrast, single-model baselines often generate plots and text in a more opportunistic, step-by-step manner, so narrative flow, figure selection, and captioning drift away from the original task specification, lowering both readability and visual quality, despite the successful execution of some individual steps.

Summarized Answer to Q1. Taken together, the improvements in Report Quality, Completeness, Scientific Rigor, Readability, and Visual Quality demonstrate that coordinated task planning is a key determinant of end-to-end performance on Climate-Agent-Bench-85. By explicitly decomposing workflows and distributing responsibilities across specialized agents, ClimateAgent produces higher-quality scientific reports and more complete end-to-end workflows than strong GPT-5–based single-model baselines. These quantitative findings support Q1, confirming that collaborative division of labor among specialized agents yields systematic gains over advanced single-model reasoning with execution validation.

5.3 Experimental Results about Context Coordination

To answer Q2, we complement the quantitative report scores with a qualitative analysis of end-to-end workflows. Because each report is the outcome of a multi-step climate analysis pipeline, any miscoordination between stages (e.g., inconsistent data sources, mismatched parameter choices, or failed intermediate steps) directly degrades report quality or even prevents report generation altogether. While the scores in §5.2 show how well each system performs overall, they do not reveal why a system succeeds or fails. By qualitatively examining generated reports and their underlying execution traces, we can assess whether each system maintains the necessary cross-step dependencies and methodological consistency that define contextual coordination. Figure 2 presents representative outputs across the six climate domains, which we use to analyze how each system maintains (or fails to maintain) contextual coordination in practice.

Experiment Results and Discussions. We enumerate the key observations and discussion below:

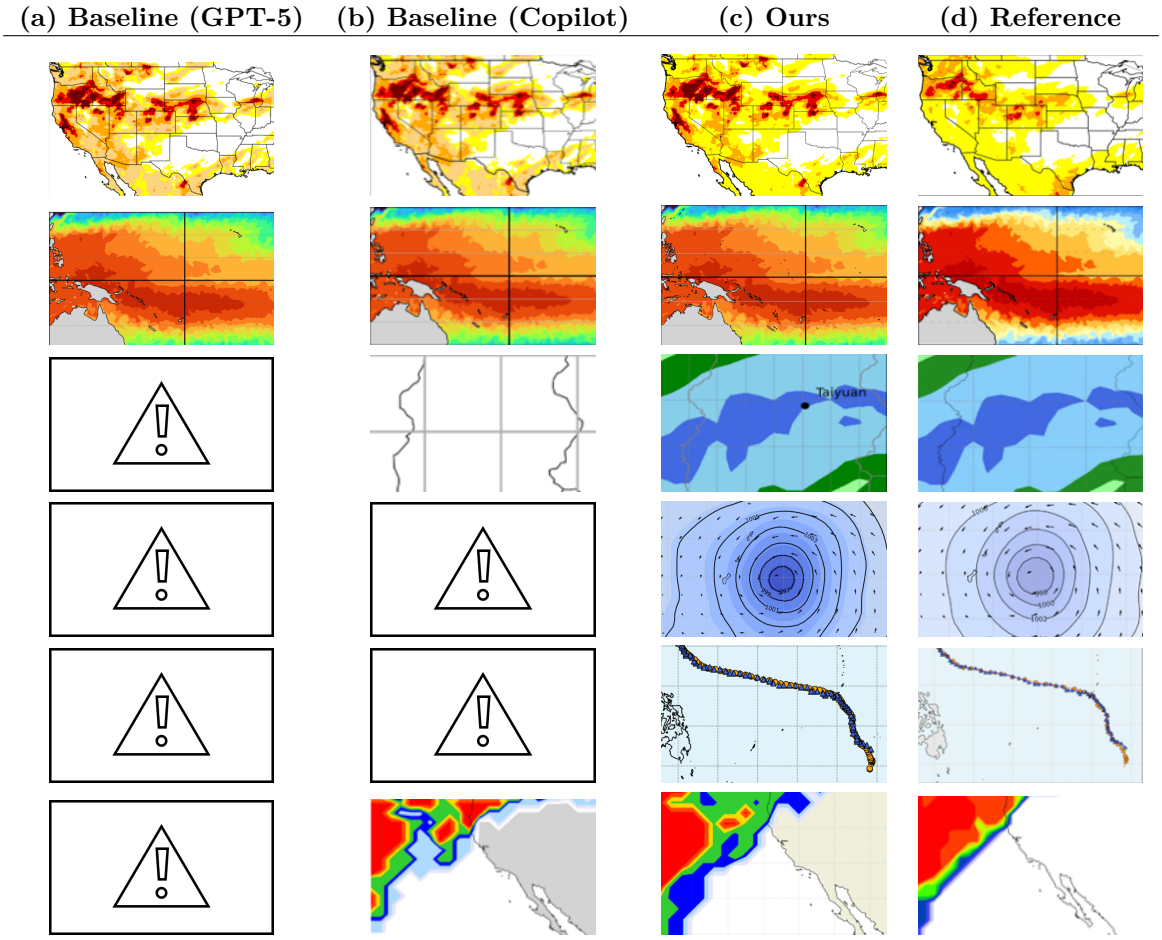


Figure 2: Qualitative comparison of generated figures for representative tasks. Each row corresponds to a climate task: (1) Drought (DR), (2) Sea Surface Temperature (SST), (3) Extreme Precipitation (EP), (4-5) Tropical Cyclone (TC), (6) Atmospheric River (AR). Columns: (a) Baseline (GPT-5), (b) Baseline (Copilot), (c) Ours, (d) Golden answer.

- **Baseline Competence in Simpler Domains.** For simpler domains such as Drought (DR) and Heatwave (HW), both baselines generally produce reasonable figures and narratives: the underlying workflows involve fewer stages, simpler data choices, and more direct mappings from task description to code. In these settings, Copilot substantially outperforms GPT-5, reflecting the benefit of iterative code refinement and execution feedback for correcting obvious bugs and filling in missing steps. However, even in these easy domains, baseline reports sometimes exhibit mild inconsistencies in variable naming, axis labeling, or temporal aggregation, foreshadowing the more severe coordination failures that emerge as workflow complexity increases. In contrast, ClimateAgent maintains a stable mapping between task specification, data selection, and visual presentation, even in these simpler cases, indicating that contextual coordination mechanisms are active across the full task spectrum.

- **Monolithic Scripts.** As workflows become more complex in Atmospheric River (AR) and Sea Surface Temperature (SST) tasks, baseline systems increasingly default to monolithic scripts with limited explicit context management. Qualitative inspection of execution traces shows a recurring pattern: changes made late in the script to fix an exception or adjust a plot overwrite earlier logic without updating dependent steps, so the final code no longer matches the original analytical intent. This leads to misaligned preprocessing, inconsistent temporal or spatial domains across figures, and missing climatological references in SST tasks. ClimateAgent, by contrast, decomposes these workflows into explicit substeps with clearly defined inputs and outputs, ensuring that updates to one stage (e.g., data filtering or anomaly com-

putation) can be propagated to downstream analyses and visualizations, thereby preserving cross-step dependencies required for contextual coordination.

- **Untracked External Processes.** In demanding domains such as Tropical Cyclone (TC) and Extreme Precipitation (EP), baseline failures are often driven by how external tools and processes are orchestrated. When GPT-5 or Copilot spawns new processes (e.g., calling TempestExtremes for TC tracking), return codes and intermediate outputs are rarely tracked rigorously. Downstream code frequently proceeds as if upstream stages had succeeded, attempting to read missing files, operate on partially written datasets, or plot fields that were never computed. This results in runtime crashes, empty or nonsensical visualizations, and incomplete reports, especially in TC tasks that rely on multi-step external toolchains. In contrast, ClimateAgent treats each external invocation as an explicit workflow step, recording return statuses, verifying that expected artifacts exist and are well-formed, and halting or repairing the pipeline when upstream tools fail, thereby maintaining coherent execution state across process boundaries.
- **Missing Intermediate Validation.** Across all domains, these issues are compounded by a pervasive lack of systematic validation of intermediate results in the baselines. Figures are often generated without checking whether the underlying data satisfy task requirements (e.g., correct spatial subset, sufficient temporal coverage, or non-degenerate statistics), leading to qualitatively poor or even empty plots when earlier steps quietly fail or return trivial outputs. A misconfigured data request or misaligned coordinate system at the beginning of the workflow can thus cascade into misleading or uninformative visualizations at the end, with no mechanism to detect or correct the deviation. ClimateAgent mitigates this cross-domain failure mode by embedding validation hooks throughout the workflow — checking the dataset metadata, asserting non-empty and physically plausible fields, and aligning coordinate systems before plotting — so that each stage both consumes and produces well-validated context, reinforcing contextual coordination from data acquisition through to final report generation.

Summarized Answer to Q2. These findings confirm that **Contextual Coordination** (Q2) is the critical differentiator for robust scientific workflows. The analysis reveals that baseline failures in complex domains are rarely due to local coding errors, but rather a structural inability to maintain state across long horizons and process boundaries. ClimateAgent overcomes this by enforcing explicit, validated handoffs between agents, effectively treating intermediate artifacts as contracts. This persistent state management ensures that downstream execution remains strictly conditioned on upstream results, preventing the context drift that causes single-model baselines to lose the analytical thread in multi-stage tasks.

5.4 Ablation and Case Studies for Adaptive Self-Correction

We now turn to answer Q3 — the system's ability to autonomously identify failures, revise its plans, and continue execution without human intervention. This section analyzes common baseline failure modes and demonstrates how ClimateAgent achieves robustness through multi-layered detection, recovery, and adaptive replanning mechanisms. To validate this capability, aggregate performance metrics are insufficient; we must instead isolate the specific mechanisms of failure and recovery. Therefore, we adopt a two-pronged approach: (1) an ablation analysis of the distribution of errors in baselines to define the failure modes our system must overcome, and (2) a case study tracing a complex recovery loop to demonstrate the self-correction mechanism in action.

Ablation Analysis. We systematically classified GPT-5 baseline errors across 35 failed tasks, identifying six primary categories (Table 3): Data/Array Shape or Key Errors (26%), Data Request Errors (17%), Syntax/Indentation Errors (11%), Timeout Errors (11%), Type Errors (11%), and Miscellaneous (23%). These failures predominantly result in incomplete or absent report generation, demonstrating direct LLM code synthesis limitations without system-level safeguards.

Our system incorporates several architectural and prompt-based interventions that directly address the failure modes observed in the baseline. Figures 3–6 illustrate how specialized Data-Agent and Coding-Agent components validate dataset metadata, enforce typing, and iteratively repair code before execution, eliminating the majority of such errors. We summarize the key findings below:

Table 3: Summary of Error Categories and Their Counts

Error Category	Count
Data/Array Shape or Key Error	9
Data Request Error	6
Syntax/Indentation Error	4
Timeout Error	4
Type Error	4
Miscellaneous	8

Baseline GPT-5 Error Example

```
# Incorrect boolean mask shape
ivt_selected = ivt[:, mask]
```

Our System Success

```
# Ensures mask shape matches data shape
mask = (ivt > threshold)
# 1D selection
ivt_selected = ivt[mask]
```

Figure 3: Comparison of array indexing: the baseline code fails due to a shape mismatch in boolean indexing, while our system validates shapes and applies correct masking.

Baseline GPT-5 Error Example

```
# Incorrect date range format
# in API request
request = {
    ...
    "date": f"{req_start}/to/{req_end}",
    ...
}
```

Our System Success

```
# Validates date range for each month
request = {
    "year": str(year),
    "month": month,
    "day": days,
}
```

Figure 4: Comparison of ERA5 data request formatting: the baseline code fails due to an invalid date range string, while our system programmatically generates and validates correct request parameters, preventing API errors.

Baseline GPT-5 Error Example

```
# Syntax error: unmatched '}'
dtxt += f} {bearing}"
```

Our System Success

```
# No syntax or indentation errors.
```

Figure 5: Comparison of syntax handling: the baseline code fails with a syntax error due to an unmatched brace, while our system's coding agent ensures only syntactically valid code is executed.

- **Data/Array Shape or Key Error:** As illustrated in Figure 3, baseline approaches often fail due to incorrect assumptions about data structure or mismatched array dimensions. Our system addresses these issues: Data-Agents extract and validate dataset metadata before code generation, ensuring that only available variables and correctly shaped dimensions are used. Furthermore, Coding-Agents perform LLM-based code validation to verify data access patterns before execution, systematically preventing such shape and key errors.
- **Data Request Error:** As illustrated in Figure 4, baseline approaches fail when interacting with ERA5 or ECMWF APIs, often due to improperly formatted requests or insufficient parameter validation. Our Data-Agents employ automated metadata extraction and validation routines before code generation,

Baseline GPT-5 Error Example

```
# TypeError: not all arguments
# converted during string formatting
df["LON"].apply(wrap_lon_to_360)
```

Our System Success

```
# Convert LON to numeric
# before modulo operation
pd.to_numeric(df_filt["LON"])
df_filt["LON"] % 360
```

Figure 6: Comparison of longitude alignment: the baseline code fails when non-numeric values are present, while our system ensures type safety before applying arithmetic operations.

utilizing LLM-driven logic to dynamically select valid variables, date ranges, and request options, referencing up-to-date dataset metadata to construct compliant API calls.

- **Syntax/Indentation Error:** Figure 5 highlights how baseline LLM-generated code frequently encounters syntax and indentation problems that halt execution. Our Coding-Agents proactively check for such errors before running any code, using diagnostic feedback to iteratively refine and correct the code, ensuring that only error-free scripts proceed to execution.
- **Type Error:** As demonstrated in Figure 6, baseline code often fails due to improper handling of data types. Our system integrates type validation and conversion directly into the workflow, with the Coding-Agent automatically checking and enforcing correct data types — guided by both prompt instructions and LLM-based code review — before any computation is performed.

Case Study. The tropical-cyclone task focusing on Typhoon Noru (SID: 2022264N17132) demonstrates the full value of our robustness mechanisms. This task compares the observed historical track from the IBTrACS dataset against a simulated track generated using ERA5 reanalysis data, producing a meteorological map visualizing both tracks alongside a summary report quantifying track differences and forecast accuracy.

The baseline GPT-5 code fails to complete the workflow, terminating with:

ERROR: Failed to select longest track. single positional indexer is out-of-bounds

This failure is due to incomplete or improperly parsed track data.

In contrast, our system executes a robust, agent-based workflow:

- Plan-Agent: Decomposes the TC analysis into 10 explicit subtasks, including: (1) reading and processing IBTrACS data, (2) determining ERA5 download parameters, (3) downloading ERA5 reanalysis data, (4) computing TempestExtremes detection parameters, (5) running DetectNodes, (6) running StitchNodes, (7) extracting the longest simulated track, (8) visualizing meteorological fields, (9) extracting central pressure, and (10) generating the final Markdown report.
- Data-Agent: Utilizes metadata extraction and LLM-guided parameter selection to construct valid ERA5 API requests. The agent automatically identifies the correct dataset (`reanalysis-era5-single-levels`) and validates all required parameters (such as date ranges and area) before making the API call.
- Coding-Agent: Implements robust scripts for each downstream subtask, including dynamic parameter computation, file validation, and error handling. Each script checks for the existence and integrity of its outputs before passing control to the next step, ensuring that failures are caught early and reported with actionable diagnostics. Finally, the agent compiles the Markdown report, embedding the meteorological field plot and central pressure value.

Summarized Answer to Q3. These results substantiate **Adaptive Self-Correction (Q3)**. The proper handling of errors, combined with the qualitative success in the Typhoon Noru case study, demonstrates that robustness in scientific agents cannot be achieved by LLM reasoning alone. Instead, it requires a system architecture that treats code generation as a hypothesis to be validated — using metadata constraints, static analysis, and execution feedback to autonomously correct the "hallucinations" that otherwise break long-horizon scientific workflows.

6 Conclusion

We have introduced ClimateAgent, an autonomous multi-agent system designed to orchestrate complex climate science workflows from high-level user prompts to comprehensive scientific reports. By leveraging a hierarchical architecture of specialized LLM-based agents, i.e., each responsible for planning, data acquisition, analysis, and visualization, our system addresses the limitations of generic code-generation models and static scripting approaches. Extensive evaluation on the Climate-Agent-Bench-85 benchmark demonstrates that ClimateAgent substantially outperforms advanced single-model baselines across all domains, particularly in tasks requiring multi-step reasoning, robust error handling, and domain-specific knowledge. Our results highlight the effectiveness of modular agent specialization, dynamic error recovery, and context-aware orchestration in enabling reliable, end-to-end automation of climate research workflows. This work advances the state-of-the-art in scientific workflow automation and paves the way for more accessible, efficient, and reproducible climate science.

References

Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler: An extensible system for design and execution of scientific workflows. *Scientific and Statistical Database Management*, pp. 423–424, 2004.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, et al. Program synthesis with large language models, 2021.

Rasmus Benestad, Kajsa Parding, Andreas Dobler, and Abdelkader Mezghani. A strategy to effectively make use of large volumes of climate data for climate change adaptation. *Climate Services*, 6:48–54, 2017.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024. doi: 10.1038/s42256-024-00832-8. URL `https://doi.org/10.1038/s42256-024-00832-8`.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Roberto Buizza, Paul Poli, Michel Rixen, Magdalena Alonso-Balmaseda, Michael Bosilovich, Stefan Brönnimann, Gilbert Compo, Dick Dee, Franco Desiato, et al. Advancing global and regional reanalyses. *Bulletin of the American Meteorological Society*, 99(8):ES139 – ES144, 2018.

CDS. Copernicus climate data store. `https://cds.climate.copernicus.eu/`. Accessed: 2025-09-28.

Harrison Chase. Langchain: Building applications with large language models through composability. `https://github.com/langchain/langchain`, 2023. Accessed: 2025-08-31.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, et al. Evaluating large language models trained on code, 2021.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Representation Learning*, volume 2024, pp. 20094–20136, 2024.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, et al. Palm: scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24(1), 2023. ISSN 1532-4435.

Copernicus Climate Change Service (C3S). cdsapi: A python library to retrieve data from the climate data store. `https://pypi.org/project/cdsapi/`, 2019. Accessed: 2025-08-31.

CrewAI. Crewai: Framework for multi-agent collaboration with llms. https://crewai.dev, 2023. Accessed: 2025-08-31.

Ewa Deelman, Dennis Gannon, Matthew Shields, and Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25:528–540, 2009.

ECMWF. European centre for medium-range weather forecasts. https://www.ecmwf.int/. Accessed: 2025-09-28.

European Centre for Medium-Range Weather Forecasts (ECMWF). ecmwf-api-client: Ecmwf data access python client, 2022. URL https://github.com/ecmwf/ecmwf-api-client. Accessed: 2025-08-31.

David Fuchs, Steven C. Sherwood, Abhnil Prasad, Kirill Trapeznikov, and Jim Gimlett. TorchClim v1.0: a deep-learning plugin for climate model physics. *Geoscientific model development*, 17(14):5459–5475, 2024. doi: 10.5194/gmd-17-5459-2024. URL https://doi.org/10.5194/gmd-17-5459-2024.

Dawei Gao, Zitao Zeng, Weirui Shi, Xuchen Liu, Wenjie Kuang, Xin Lan, Zhijian Chen, Zheng Zhang, Zhaoxuan Dong, Wenhao Zhang, Yuzhe Chen, Xian Li, Suncong Li, Chuang Yang, Junyuan Wang, Jinlin Li, Jiahui Qiu, Rui Hou, Hongxuan Chen, Jie Tang, and Hongming Zhang. AgentScope: A flexible yet robust multi-agent platform, 2024.

Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Rishav Hada, Varun Gumma, Adrian de Wynter, Harshita Diddee, Mohamed Ahmed, Monojit Choudhury, Kalika Bali, and Sunayana Sitaram. Are large language model-based evaluators the solution to scaling up multilingual evaluation? In Yvette Graham and Matthew Purver (eds.), *Findings of the Association for Computational Linguistics: EACL 2024*, pp. 1051–1070, St. Julian's, Malta, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-eacl.71/.

Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, et al. The era5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020.

Stephan Hoyer and Joseph J. Hamman. xarray: N-d labeled arrays and datasets in python. *Journal of Open Research Software*, 5:10, 2017.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? A Critical Survey of Self-Correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. doi: 10.1162/tacl_a_00713. URL https://doi.org/10.1162/tacl_a_00713.

Ross King, Jem Rowland, Stephen Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, et al. The automation of science. *Science*, 324:85–89, 2009.

Qintong Li, Leyang Cui, Lingpeng Kong, and Wei Bi. Exploring the reliability of large language models as customized evaluators for diverse NLP tasks. In Owen Rambow, Leo Wanner, Marianna Apidianaki, Hend Al-Khalifa, Barbara Di Eugenio, and Steven Schockaert (eds.), *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 10325–10344, Abu Dhabi, UAE, 2025. Association for Computational Linguistics. URL https://aclanthology.org/2025.coling-main.688/.

Hilary J Oliver, Matthew Shin, and Oliver Sanders. CYLC: a workflow engine for cycling systems. *The Journal of Open Source Software*, 3(27):737, 2018. doi: 10.21105/joss.00737.

Jonathan Overpeck, Gerald Meehl, Sandrine Bony, and David Easterling. Climate data challenges in the 21st century. *Science*, 331:700–702, 2011.

Mengxin Pan and Mengqian Lu. A novel atmospheric river identification algorithm. *Water Resources Research*, 55(7):6069–6087, 2019. doi: 10.1029/2018WR024407. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR024407.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

HoloViz Project. Earthml: Machine learning and analysis tools for earth science, 2020. URL https://earthml.holoviz.org. Accessed: 2025-08-31.

Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, and F Prabhat. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

Thomas Richards. Auto-gpt: An autonomous gpt-4 experiment, 2023. URL https://github.com/Significant-Gravitas/Auto-GPT. Accessed: 2025-08-31.

Mattia Righi, Bouwe Andela, Veronika Eyring, Axel Lauer, Valeriu Predoi, Manuel Schlund, Javier Vegas-Regidor, Lisa Bock, Björn Brötz, et al. Earth System Model Evaluation Tool (ESMValTool) v2.0 – technical overview. *Geoscientific model development*, 13(3):1179–1199, 2020. doi: 10.5194/gmd-13-1179-2020.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Advances in Neural Information Processing Systems, Red Hook, NY, USA, 2023. Curran Associates Inc.

Akshay Sridhar, Zeyuan Huang, Weimin Chen, Alistar White, Tom Beucler, Trevor James Phillips, Stephan Rasp, Venkatramani Balaji, Elizabeth A. Barnes, Timothy A. O'Brien, Oliver Fuhrer, Pierre Gentine, and Christopher S. Bretherton. ClimSim-Online: A large multi-scale dataset and framework for hybrid physics-machine learning climate modeling, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, et al. Llama 2: Open foundation and fine-tuned chat models, 2023.

Paul A. Ullrich, Colin M. Zarzycki, Elizabeth E. McClenny, Marielle C. Pinheiro, Alyssa M. Stansfield, and Kevin A. Reed. Tempestextremes v2.1: a community framework for feature detection, tracking, and analysis in large datasets. *Geosci. Model Dev.*, 14:5023–5048, 2021. doi: 10.5194/gmd-14-5023-2021.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7534–7550. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.609.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6), 2024. doi: 10.1007/s11704-024-40231-1.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Hao Xue and Flora D. Salim. PromptCast: A new prompt-based learning paradigm for time series forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 36:6851–6864, 2022. doi: 10.1109/TKDE.2022.3220067.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2022.

Dan Zhang, Sining Zhoubian, Min Cai, Fengzu Li, Lekang Yang, Wei Wang, Tianjiao Dong, Ziniu Hu, Jie Tang, and Yisong Yue. Datascibench: An llm agent benchmark for data science. *arXiv*, 2502.13897, 2025a. URL https://arxiv.org/abs/2502.13897.

Xingjian Zhang, Yutong Xie, Jin Huang, Jinge Ma, Zhaoying Pan, Qijia Liu, Ziyang Xiong, Tolga Ergen, Dongsub Shim, Honglak Lee, and Qiaozhu Mei. Massw: A new dataset and benchmark tasks for ai-assisted scientific workflows. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2373–2394, Albuquerque, New Mexico, 2025b. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.127. URL https://aclanthology.org/2025.findings-naacl.127/.

Yifan Zhang, Yang Yuan, and Andrew Chi-Chih Yao. Meta prompting for ai systems, 2025c. URL https://arxiv.org/abs/2311.11482.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

A Appendix

A.1 Plan-Agent: Task Decomposition and Agent Assignment

The Plan-Agent is responsible for decomposing high-level user tasks into a sequence of actionable subtasks, each assigned to a specialized agent within the ClimateAgent system. Leveraging large language models, the Plan-Agent generates detailed, chronologically ordered plans that preserve all scientific and technical requirements from the original user prompt. Each subtask includes explicit instructions, file paths, parameter values, and workflow conventions to ensure downstream agents can execute their roles unambiguously and reproducibly.

Prompt Engineering. The Plan-Agent constructs prompts that enumerate available agents (data download, programming, visualization), file system conventions, and dataset constraints. It instructs the LLM to retain all relevant details from the user task, specify agent assignments, and output the plan as a structured JSON list. This approach ensures that each subtask is both specific and actionable, minimizing ambiguity and error propagation.

Plan-Agent LLM Prompt

You are a planning agent for a modular climate forecasting/reporting system. The system has four main agents:

- `cdsapi_download_agent`: downloads climate data only using the cdsapi library (for Copernicus Data Store datasets).
- `data_download_agent`: downloads climate data only using ecmwf-api-client library (for ECMWF S2S dataset only).
- `programming_agent`: processes, analyzes, and computes on climate data (but does not download data). It should generate plots/graphs that will be used in the final report.
- `visualization_agent`: generates the final report in Markdown format, including all requested plots, visualizations, and human-friendly interpretation. The final report generation should always be handled by the visualization_agent.

File System & Data Conventions

- Only download agents may write files to `data/`.
- Coding-Agents must write all outputs (processed data, intermediate results, analysis outputs, figures, etc.) to `code_output/`.
- Visualization agents should read from `code_output/`.
- All user-provided data must be referenced under `../user_provided_data/` (relative to the task root).

Critical: When generating subtasks, you MUST preserve as much detail as possible from the original user task prompt. For each subtask, explicitly include all relevant parameters, file paths, scientific logic, and requirements from the original prompt. Do not summarize or omit details. If the original prompt specifies variables, thresholds, file formats, coordinate conventions, or workflow steps, these MUST be included in the subtask description. The goal is for each subtask to be as actionable and unambiguous as possible for downstream agents.

Planning Guidelines

- Break down the main task into small, logical, and chronologically ordered steps.
- For each subtask, specify the agent, the action, and all required details (parameters, files, scientific logic, etc.).

- Be specific and unambiguous: name required data sources, variables, parameters, and outputs precisely.
- If a subtask involves generating a plot/graph, specify the type of plot and the data to be used.
- If a subtask is for the visualization_agent, make sure the description clearly states that it is for the final report and should include all required plots and interpretation.
- Only include agent types relevant to the task (do not include unused agent types).

Example of Generated Plan. Below is an example of a plan generated by the PLAN-AGENT for an atmospheric river (AR) detection workflow. Each subtask is assigned to the appropriate agent and includes a detailed description of the required actions, parameters, and output conventions.

Plan-Agent Output Example

Subtasks:

1. **Create configuration file and directory structure for AR detection workflow**
   Agent: programming_agent
   Create `code_output/config.py` with constants and paths, and ensure all directories exist. Write `run_metadata.json` in `code_output/outputs_ar_freq` with the configuration used.
2. **Download ERA5 pressure-level q, u, v for 2022-12-19 to 2022-12-25 at 00:00 UTC**
   Agent: cdsapi_download_agent
   Use cdsapi to request 'reanalysis-era5-pressure-levels' data for specific humidity, u and v wind components, at specified pressure levels and times. Save output to `data/era5_raw/era5_pl_uvq_20221219-20221225_0000.grib`.
3. **Compute daily IVT on a 1.5° x 1.5° grid for each date in DATE_RANGE**
   Agent: programming_agent
   Script: `code_output/compute_ivt.py`. Load downloaded data, regrid to 1.5° grid, compute IVT, and save daily NetCDF files with compression.
4. **Load monthly IVT threshold and run AR detection per day (extract, rotate, paths, trajectories, metrics)**
   Agent: programming_agent
   Script: `code_output/detect_ar_pathways.py`. Load threshold, apply rotation, compute anomaly, extract paths, generate trajectories, and save outputs in hierarchical date folders.
5. **Weekly aggregation of daily pathway presence and create frequency map image**
   Agent: programming_agent
   Script: `code_output/visualize_frequency.py`. Aggregate daily pathway files, compute weekly frequency, subset to US West Coast, save NetCDF, and plot frequency map using Cartopy.
6. **Generate final Markdown report with the exact reference text and embedded image**
   Agent: visualization_agent
   Create `code_output/report.md` referencing the frequency map image and containing the required summary and bullet points.

A.2 Data-Agents: Prompt Engineering and Metadata Extraction

The DATA-AGENTS in CLIMATEAGENT, namely ECMWF-AGENT and CDSAPI-AGENT, are responsible for generating robust Python scripts to download climate datasets from ECMWF S2S and Copernicus Data Store (CDS), respectively. These agents leverage large language models (LLMs) for code synthesis and employ automated metadata extraction using browser automation to ensure parameter validity and dataset availability.

Prompt Engineering. Both agents utilize carefully designed prompts to instruct the LLM in generating executable download scripts. The prompts include explicit requirements for code structure, error handling, output directory usage, and metadata integration.

ECMWF-Agent Prompt Example. We attach the agent prompt for ECMWF below:

ECMWF-Agent LLM Prompt

You are an expert in ECMWF S2S data download. Given the following task, parameter info, and metadata JSONs, write a Python script that downloads the required data using the ecmwf-api-client library.
- Use only the available options in the metadata JSONs (see file names and their content summaries below).
- The purpose of these metadata JSONs is to reduce errors in the API calls of the generated Python code.

General Requirements for the Code:

- The code must be modular, well-structured, and include clear, descriptive comments explaining each step and function.
- Follow Python best practices for readability, maintainability, and efficiency.
- Use appropriate scientific/data libraries (e.g., numpy, pandas, matplotlib, xarray, etc.).
- All necessary imports must be included at the top of the script.
- All code should be directly executable, with all necessary fields and values filled in.
- The script must be runnable as: python [code_name].py (with no arguments). Do not require or parse any command-line arguments.
- All downloaded data files must be saved in the 'data' subdirectory of the current task folder. Do not save data files elsewhere.
- Do NOT use 'experiments/user_provided_data' for any downloaded data outputs. That directory is reserved for user-provided data only.
- All relative paths should be constructed relative to the directory the code is running. Don't use absolute paths.

IMPORTANT: For efficiency, always batch all required parameters and levels into a single API call using '/'-separated lists for 'param' and 'levelist'. Do NOT make a separate retrieve call for each parameter or level unless required by the API. Only loop over forecast type (cf/pf) or origin if absolutely necessary.

Available parameters (with codes): `{AVAILABLE_PARAMS_TEXT}`

IMPORTANT: Always use the correct 'origin' code for the requested model/database.

Special Instructions for Data Downloading:

- If the subtask involves data download, you must use the `ecmwf-api-client` library or the provided download tool.
- Only include the required fields below in the API call (do not add any others, especially 'area'):
  - class: s2
  - dataset: s2s
  - date: <date range for real-time|model version date for hindcast>
  - expver: prod
  - levelist: <level range> (only for pl, omit for sfc)
  - levtype: <sfc|pl> (if requesting both, call API separately for each)
  - model: glob
  - origin: <origin> (e.g. anso, ecmf, kwbc)
  - param: <parameter> (if requesting multiple parameters, call API separately for each)
  - step: <step range> (use a '/'-separated list of step values)
  - stream: <enfo|enfh> (enfo: realtime, enfh: hindcast)
  - time: '00:00:00'
  - type: <cf|pf> (cf: control, pf: perturbed)
  - target: <target file name>
  - hdate: <yyyy-mm-dd> (only for hindcast, specify a list of hindcast initialization dates)
  - number: <number of ensemble members> (only for perturbed hindcast, use '/'-separated list for multiple members)
- Do not include any other fields.

Guideline for Setting 'date', 'hdate', and 'step' Fields in Requests:

1. Real-Time Forecast Setting: Use the operational model version available on the date.
2. Hindcast (Reforecast) Setting: Use the most recent available model version date strictly before the requested date.
3. 'step' field: For daily-averaged parameters, use hour ranges representing 24-hour periods; for instantaneous/accumulated parameters, use single time steps.

After Downloading the Data:

- Create/Update a README.md file in the data directory, listing all downloaded files and their descriptions.
- For GRIB files, use `cfgrib` to extract and include metadata summaries in the README.
- For other file types, provide appropriate previews or summaries.

Metadata JSONs (file name, description, and content preview): `{meta_block}`

Task description: `{task_description}`

Return only the Python code, with all explanations and context provided through code comments. Do not include any narrative or markdown outside the code block.

CDSAPI-Agent Prompt Example: We attach the agent prompt for CDSAPI below:

CDSAPI-Agent LLM Prompt

You are an expert in CDS (Copernicus Data Store) data download. Given the following task, write a Python script that downloads the required data using the cdsapi library.

- The code must be modular, well-structured, and include clear, descriptive comments explaining each step and function.
- Use only the required fields for the cdsapi call (see https://cds.climate.copernicus.eu/api-how-to for reference).
- All necessary imports must be included at the top of the script.
- All code should be directly executable.
- The script must be runnable as: python [code_name].py (with no arguments). Do not require or parse any command-line arguments.
- All downloaded data files and the README.md must be saved in the directory: [DATA_DIR], which will be the current working directory when the script is run.
- The script must use the current working directory (`os.getcwd()`) or a provided variable for all output paths.
- After downloading, create or update a README.md file in the data directory, listing the files and a brief description of their contents.
- If the dataset or variable is not available, the script should print a clear error message.
- Note that the `cdsapi.Client` only supports the `retrieve` method.
- At the end of the script, print to stdout a single line containing a JSON array of the absolute paths of all files that were downloaded by the script. For example: `print(json.dumps(["/path/to/file1", "/path/to/file2"]))`
- Do not print anything else to stdout after this line.

Task description: `{task_description}`

Metadata: `{metadata_str}`

Return only the Python code, with all explanations and context provided through code comments. Do not include any narrative or markdown outside the code block.

Metadata Extraction via Chrome/Selenium. To dynamically identify available datasets, variables, and valid parameter ranges, both agents use Selenium with Chrome in headless mode. The agent navigates to the relevant data portal, interacts with web forms, and parses metadata (e.g., from JavaScript objects or HTML elements). Extracted metadata is saved as JSON and provided to the LLM as context for code generation, ensuring that only valid options are used in download requests.

Outputs. The agents produce several outputs for each download task:

- **Generated Python Script:** A modular, well-commented script that downloads the requested data using validated parameters.
- **README.md:** A summary file listing downloaded files and their descriptions.
- **Metadata JSON:** A record of available dataset options and parameters.

Example: README.md file

ERA5 Pressure-Level Data Raw Download
Dataset: reanalysis-era5-pressure-levels
Variables: specific_humidity, u_component_of_wind, v_component_of_wind
Pressure levels (hPa): 1000, 925, 850, 700, 500, 300, 200
Time: 00:00 UTC
Date range: 2022-03-24 through 2022-03-30
Format: GRIB
File path: `data/era5_raw/era5_pl_uvq_20220324-20220330_0000.grib`

These data are native ERA5 pressure-level fields suitable for IVT computation.

Discussion. By combining LLM-driven code generation with automated metadata extraction, the DataAgents reduce errors due to invalid parameters and improve reproducibility. This approach enables the system to adapt to evolving data portals and ensures that download scripts remain robust and up-to-date.

A.3 Coding-Agent (Programming): Data Processing and Analysis

The Coding-Agent (Programming) is responsible for generating Python code to perform analysis and processing subtasks within the ClimateAgent workflow. This agent leverages large language models (LLMs) to synthesize modular, well-documented scripts for scientific data analysis and post-processing, based on the main user task and specific subtask descriptions.

Prompt Engineering. The agent constructs detailed prompts for the LLM, specifying requirements such as code modularity, use of scientific libraries (e.g., numpy, pandas, xarray, matplotlib), and strict file system conventions. The prompt instructs the LLM to avoid data download operations (handled by dedicated agents), save all outputs in the `code_output/` directory, and update or create a `README.md` file describing generated outputs. Debugging instructions and error messages are included in the prompt when code regeneration is required.

Coding-Agent (Programming) LLM Prompt

You are an expert Python programmer and agent developer. You are programming for predicting and forecasting atmospheric phenomena in climatology. Downloads from ECMWF datasets are already handled by the Data-Agents. Do not write code to download climate data.
General Requirements for the Code:

- The code must be modular, well-structured, and include clear, descriptive comments explaining each step and function.
- Follow Python best practices for readability, maintainability, and efficiency.
- Use appropriate scientific/data libraries (e.g., numpy, pandas, matplotlib, xarray, etc.).
- All necessary imports must be included at the top of the script.
- All code should be directly executable.
- All output files must be saved in the `code_output/` directory under the current task root.
- Do not write any files to the `data/` directory.
- All user-provided data must be loaded from the directory `../user_provided_data/`.
- For every output file generated, also create or update a `README.md` file in the output directory describing the outputs.
- Plotting tip: Always place your colorbar or legend outside the main plot area and use `tight_layout` or `constrained_layout` for spacing.

Write a Python script that executes the given **subtask**. At the end of the script, include a test script to validate the generated output. Return only the Python code, with all explanations and context provided through code comments. Do not include any narrative or markdown outside the code block.
Main task: `{main_task}`
Subtask: `{subtask}`
Previous subtasks and codes: `{previous_codes}`
Directory structure: `{dir_tree}`
README.md summary: `{readme_summary}`

Workflow and Error Recovery. For each subtask, the agent generates multiple candidate scripts, validates their syntax, and ranks them using LLM-based code review for correctness, robustness, and clarity. If all candidates fail, the agent enters a debug loop, providing error messages and previous code to the

LLM for iterative refinement. Successful code execution triggers automatic updates to the `README.md` file, documenting outputs and code changes.

Output. The following is an example of a `README.md` file automatically generated by the Coding-Agent (Programming) after completing a subtask for weekly sea surface temperature (SST) analysis. This file documents the produced figures and reports, describes the contents and projection details, and provides metadata for reproducibility and further analysis.

Example: README.md for Analysis Output

Weekly SST and SST Anomaly Map
This directory contains the high-resolution two-panel map showing:

- Weekly mean Sea Surface Temperature (SST) for June 12–18, 1997
- Weekly mean SST Anomalies for the same period

File:

- `weekly_sst_anomaly_map.png`: Two-panel PNG figure [12×8 in, 300 DPI]

The map uses a PlateCarree projection (central_longitude = $-155°$, extent 120°E–290°E, $\pm 30°$ latitude), with bold black lines at the equator and 180° meridian, and coastlines/land shaded in gray.

Discussion. By automating code generation, validation, and debugging, the programming agent streamlines scientific analysis and ensures reproducibility. Its design enforces strict conventions for output management and documentation, facilitating transparent and collaborative climate research workflows.

A.4 Coding-Agent (Visualization): Report Generation and Plotting

The Coding-Agent (Visualization) automates the creation of scientific reports and visualizations as the final step in the ClimateAgent workflow. This agent synthesizes Markdown documents, figures, and summary files by leveraging large language models to interpret analysis outputs and generate publication-quality content. It ensures that all results are saved in standardized directories and that each output is accompanied by a descriptive `README.md` file for reproducibility.

Prompt Engineering. The visualization agent constructs prompts that specify the required report structure, output file conventions, and documentation standards. The prompt instructs the LLM to:

- Generate all requested plots and Markdown content as specified in the subtask.
- Save all outputs (figures, CSVs, Markdown) in the `code_output/` directory.
- For every output file, create or update a `README.md` in its directory describing the outputs.
- Ensure the final report is a Markdown file named `final_report.md` in `code_output/`, containing all relevant analysis and images.
- Load any user-provided data from the standardized directory `experiments/user_provided_data/`.
- Return only the Python code, with all explanations and context provided through code comments, and no extraneous markdown or narrative.

Coding-Agent (Visualization) LLM Prompt

You are responsible for fully executing the following visualization/reporting subtask. Generate all required plots, Markdown, and outputs as specified, and ensure all results are saved and documented as described below.
Requirements:

- Assume code is run from the task root (`[output_dir]`).
- Save all outputs (figures, CSVs, Markdown) in `code_output/`.
- For every output file, create or update a `README.md` in its directory describing the outputs.
- The final output must be a Markdown report named `final_report.md` in `code_output/`, containing all relevant analysis and images.
- All user-provided data must be loaded from `experiments/user_provided_data/`.
- Return only the Python code, with all explanations and context provided through code comments. Do NOT include any narrative, markdown, or code block markers outside the code.

Subtask to execute: `{subtask}`
Main Task: `{main_task}`
Context: Directory structure, previous subtasks, previous codes, and key README.md summaries.

Workflow and Error Recovery. For each visualization subtask, the agent generates multiple candidate scripts, validates their execution, and iteratively refines code in response to errors. It gathers context from previous analysis outputs, directory structure, and documentation to ensure consistency and completeness in the final report.

Output Examples. Below are two examples of `final_report.md` files automatically generated by the Coding-Agent (Visualization) after completing different visualization subtasks. These reports demonstrate the agent's versatility in synthesizing diverse scientific findings — from multi-day precipitation events to tropical cyclone meteorological analyses — into publication-ready markdown documents with embedded figures, spatial analysis, and narrative summaries for transparent communication and reproducibility.

Example: final_report.md for Extreme Precipitation Event

Extreme Precipitation Event in the Greater Bay Area (2023-09-05 to 2023-09-10)

Introduction

This report summarizes an exceptional precipitation event that affected the China Greater Bay Area between September 5 and 10, 2023. Hourly total precipitation (`tp`) data were obtained from the ERA5 reanalysis via the Copernicus Climate Data Store and aggregated to daily totals (mm). Spatial analysis and visualization were performed on the native ERA5 grid, highlighting regions exceeding common thresholds (25, 50, 100, 250 mm) and tracking the evolution of rainfall cores.

Multi-Panel Precipitation Map

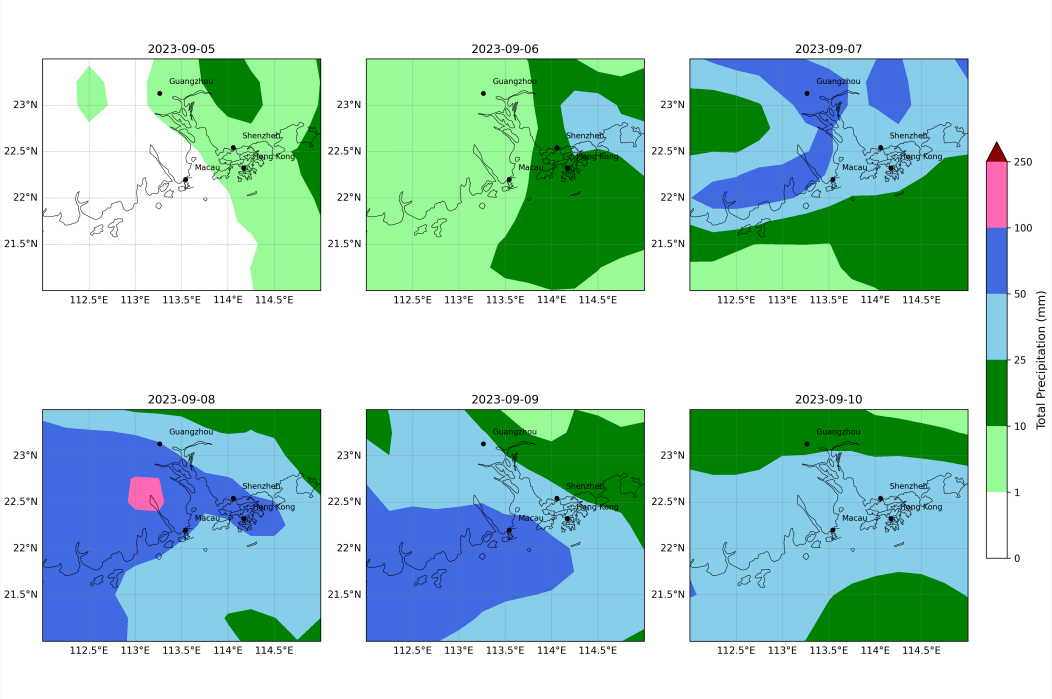


Figure 1. Daily total precipitation in the Greater Bay Area from September 5 to 10, 2023. Stepped color intervals (0, 1, 10, 25, 50, 100, 250 mm) illustrate rainfall intensity. Major cities (Guangzhou, Shenzhen, Hong Kong) are marked for reference.

Event Evolution Narrative

Onset (Sep 5–6): The event began on September 5 with scattered moderate showers (10–25 mm) over western Guangdong. By the 6th, a convergence axis intensified over inland hills, producing localized cores up to 50 mm (sky blue to royal blue shading) northeast of Guangzhou.

Intensification (Sep 7–8): On September 7, rainfall expanded eastward, with cores exceeding 100 mm over the Pearl River Delta. The 8th marked the peak growth phase: a broad swath of hot pink (100–250 mm) stretched from central Shenzhen northward, triggering flash-flood warnings along tributaries.

Peak (Sep 9): September 9 saw the maximum daily accumulation, with dark red (>250 mm) cells persisting near coastal hills. Flood impacts were reported in suburban Foshan and low-lying areas of Dongguan, where drainage systems were overwhelmed.

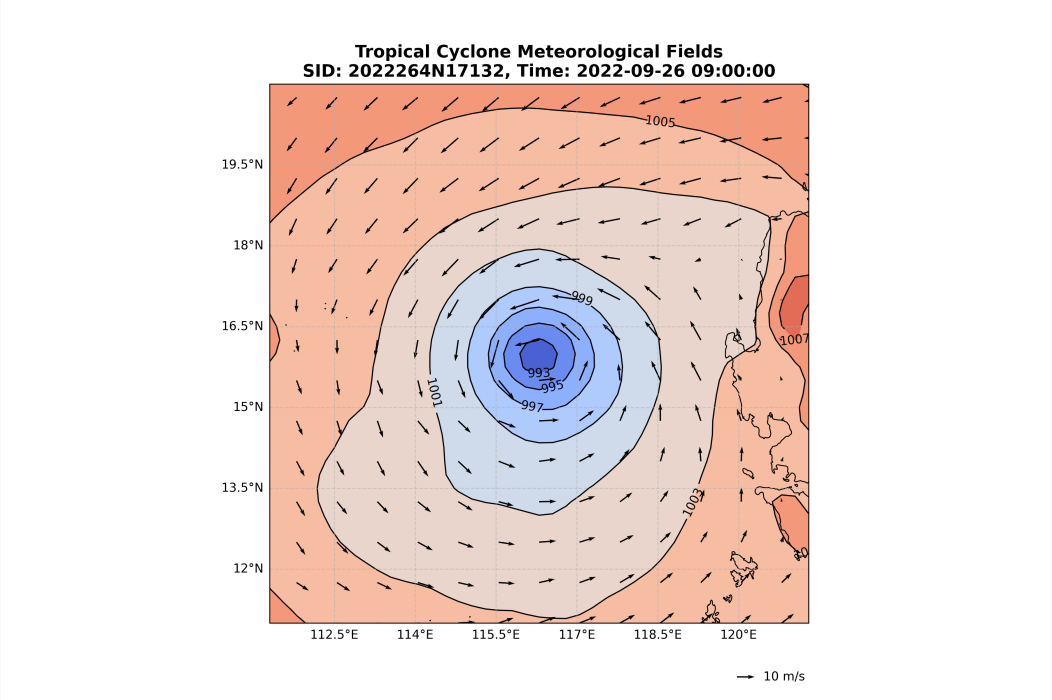
Weakening (Sep 10): By the final day, precipitation waned and shifted southward, contracting to 25–50 mm bands around Hong Kong's northern New Territories. The event concluded with isolated showers and rapid clearing.

Conclusion

This multi-day event demonstrated a classic inland-to-coastal propagation of heavy rainfall under synoptic forcing, with peak intensities exceeding 250 mm in localized cores. The spatial shift of maxima and associated flooding underscores the importance of high-resolution reanalysis for regional hazard assessment.

Example: final_report.md for Tropical Cyclone Event

Tropical Cyclone Meteorological Fields (SID: 2022264N17132)



This chart shows a low-pressure system, with a central pressure of approximately 991.8969 hPa, accompanied by a distinct counterclockwise rotating wind field. Such a cyclone may bring strong winds and heavy rainfall, requiring close monitoring of its path and intensity changes to prevent potential impacts on coastal areas or maritime activities.