Supplementary Material

Supplementary Material A1: Memory System Technical Specifications

A1.1. Vector Embedding and Similarity Search

The memory system uses Azure OpenAI's text-embedding-3-small model to generate dense semantic vectors that encode the meaning of queries and responses. During query processing, the system performs a semantic search against the memory index using a similarity threshold of 0.3. This threshold defines how closely related a stored memory must be to the current query to be retrieved, where 1.0 requires exact semantic matches and 0.0 retrieves all memories regardless of relevance.

We deliberately selected a lower threshold (0.3) than typical retrieval-augmented generation systems (which often use 0.7–0.9) to maximize conversational continuity, ensuring that loosely related prior analyses can still inform current reasoning, such as remembering a user's preferred visualization style or previously analyzed regions. A threshold of 0.9 or 1.0 would retrieve only near-exact semantic matches, causing the system to “forget” contextually relevant but differently phrased prior interactions.

Retrieved memories undergo dual filtering based on semantic similarity and metadata attributes (timestamps, variables, geographic regions), preventing irrelevant but semantically similar memories from polluting context while ensuring related prior analyses inform current reasoning.

Conversational context is recalled selectively rather than by concatenating the conversation history, so the model input does not grow with session length. Memory is injected only when a query references prior interactions; in that case, the system retrieves only memories that pass both the similarity threshold (0.3) and the metadata filter, capped at the top two most relevant memories,

which are provided together with the current request. The full dialogue history is never resent. The prompt therefore remains bounded regardless of how long a session runs, and the system relies on selective, bounded semantic recall in place of running text summarization or a fixed sliding window over raw turn.

A1.2. Memory Types and Storage

The system stores three memory types:

- User query memories preserve original natural-language requests.
- Analysis context memories record variables, regions, dates, and methodological choices, enabling comparative queries (e.g., “How does this compare to July 2023?”).
- Visualization memories store metadata about generated outputs (URLs, color scales, geographic bounds) rather than image pixels.

Memory addition occurs asynchronously after each analysis. While stored memory accumulates across a session, this storage is independent of the per-query model input, which remains bounded as described in A1.1. The detailed memory retrieval workflow is shown in Supplementary Figure A1 (Box 2).

Supplementary Figure A1. *Detailed backend execution pathway. The workflow progresses through eight processing stages: (1) API Entry & Validation, request parsing and routing with Error 400 handling; (2) Memory & Agent Selection, user ID extraction, semantic memory search (score >0.3), and agent mode selection; (3) Agent Execution, thread creation, message submission, and status polling; (4) Final Response, JSON formatting and App Insights logging; (5) Code Execution, environment setup, AST-based validation against an allow-list, and Python execution in a timeout-limited subprocess; (6) Data Processing, branching to NLDAS Kerchunk (hourly) or SPI Kerchunk (monthly) with fsspec/xarray loading; (7) Visualization, routing to static maps, overlays, animations, time series, or tiles; and (8) Storage & Response, memory embedding, blob upload, and SAS URL generation. Decision diamonds indicate validation gates; the dashed feedback loop (8→3) returns tool outputs to the agent.*

Supplementary Material A2: Backend Processing, Code Execution, and Output Specifications

API entry and error-handling detail relocated here from the former Sections 4.1.2 and 4.1.7 to keep the main text concise.

A2.1. API Entry and Request Validation

User requests enter through HTTP endpoints exposed by a FastAPI backend, which validates the incoming JSON payload to extract the natural-language query and optional spatial and temporal parameters. The model's reasoning interprets user intent and determines whether clarification is needed, so queries may be issued either as fully specified commands (e.g., “Show Tair for Florida on 2015-07-15”) or as open-ended questions (e.g., “What was the temperature like during the 2012 drought?”). Each request includes a stable pseudonymous user identifier generated locally in the browser, enabling session-specific memory retrieval without authentication or collection of personal information.

A2.2. Execution Environment

The execution environment provides pre-installed packages for data manipulation (xarray, pandas, numpy), geospatial processing (cartopy, geopandas), visualization (matplotlib), and cloud storage access (fsspec, azure-storage-blob), together with the curated library of vetted helper functions pre-loaded into the namespace.

Before execution, each generated script is parsed into an abstract syntax tree and validated against an explicit allow-list: only an approved set of scientific libraries may be imported; unsafe builtins

and introspection attributes (for example eval, exec, compile, open, and dunder or getattr/setattr access) are rejected; and reassignment of storage-credential variables is disallowed. Code that fails validation is not executed and is routed to the error-recovery path (A2.4). Validated code executes within a timeout-limited subprocess isolated from the serving process, so long-running computations are bounded by a safety termination limit rather than a single fixed duration. This constitutes static validation with process-level isolation rather than full operating-system-level sandboxing; container-level confinement and routing all data access through the helper library so that credentials are not present in the execution namespace are identified as future hardening (Section 6).

A2.3. Multi-Scale Tile Rendering

For interactive exploration, the system employs a multi-scale tile rendering strategy that adapts to the query extent:

- Small regional queries (e.g., Maryland) generate 6–8 tiles at zoom level 8.
- State-scale queries (e.g., California) produce 12–15 tiles at zoom level 7.
- Continental-scale queries (e.g., CONUS) generate 150–200 tiles at zoom levels 5–6.

Tiles are rendered as 256×256 pixel PNG images served through a custom FastAPI endpoint interfacing with the Kerchunk reference system; Azure Maps fetches them on demand as users pan and zoom.

Missing or NoData values, encoded as the CF _FillValue in the NLDAS-3 files, are decoded to NaN on read and excluded from both computation and color mapping. Each interactive tile is initialized as a fully transparent RGBA image, and only grid cells with valid data are colored, so

missing data renders as transparency rather than as a data value and the underlying base map shows through; in static maps, NaN cells are left unpainted.

A2.4. Fallback and Error Recovery

Error management operates across the pipeline. At the API layer, malformed requests are rejected with descriptive HTTP 400 responses identifying the specific validation failure. During memory retrieval, failures to reach Azure AI Search or to obtain embeddings trigger graceful degradation to stateless operation, logged while the current query proceeds without memory augmentation.

When generated code fails or contains a detectable mistake, the pre-execution repair pass corrects known mismatches and the pipeline re-runs the corrected code; failures that cannot be repaired automatically fall back to predefined templates for common query patterns. Execution failures are classified: data-loading failures return informative messages with valid alternatives, and runtime exceptions are caught and returned with guidance on simplifying the query, with internal details removed from user-facing messages. For critical failures, the system returns a minimal JSON error object containing a unique identifier, timestamp, and generic message, so the frontend always receives a parsable response.

A2.5. GeoJSON and Vector Output

GeoJSON outputs provide lightweight vector representations of analyzed data, sampling the full raster grid to manageable point densities (typically 2,000–5,000 points). Each feature includes coordinates and data values, enabling client-side filtering, styling, and interaction. The complete visualization process is detailed in Supplementary Figure A1 (Box 7).

Supplementary Material A3: Frontend Architecture and User Interface Specifications

The full User Interface Design content now lives here; the main text retains only a brief mention.

A3.1. Technology Stack

The frontend is a single-page web application built with Vite, React, and TypeScript. The complete frontend interaction workflow is illustrated in Supplementary Figure A2.

A3.2. User Interface Components

Natural Language Query Panel. Accepts domain-specific questions such as “Show temperature in Texas, May 2023,” with a scrollable history enabling follow-up prompts that reference earlier outputs.

Visualization Dashboard. Results are presented through three modes: a backend-generated static PNG with legend and color scale; interactive tile layers for large-area queries (256×256 tiles via Azure Maps); and a georeferenced transparent overlay with hover interrogation via GeoJSON. The mode is selected automatically from query characteristics.

A3.3. Session Management and Transparency Features

The header provides a New Chat function that resets memory by generating a new pseudonymous ID; a user-identity display; and an expandable Debug Info panel showing raw backend JSON (generated Python, payloads, tile parameters, memory-retrieval results), enabled in development and disabled for production.

A3.4. Backend Communication Protocol

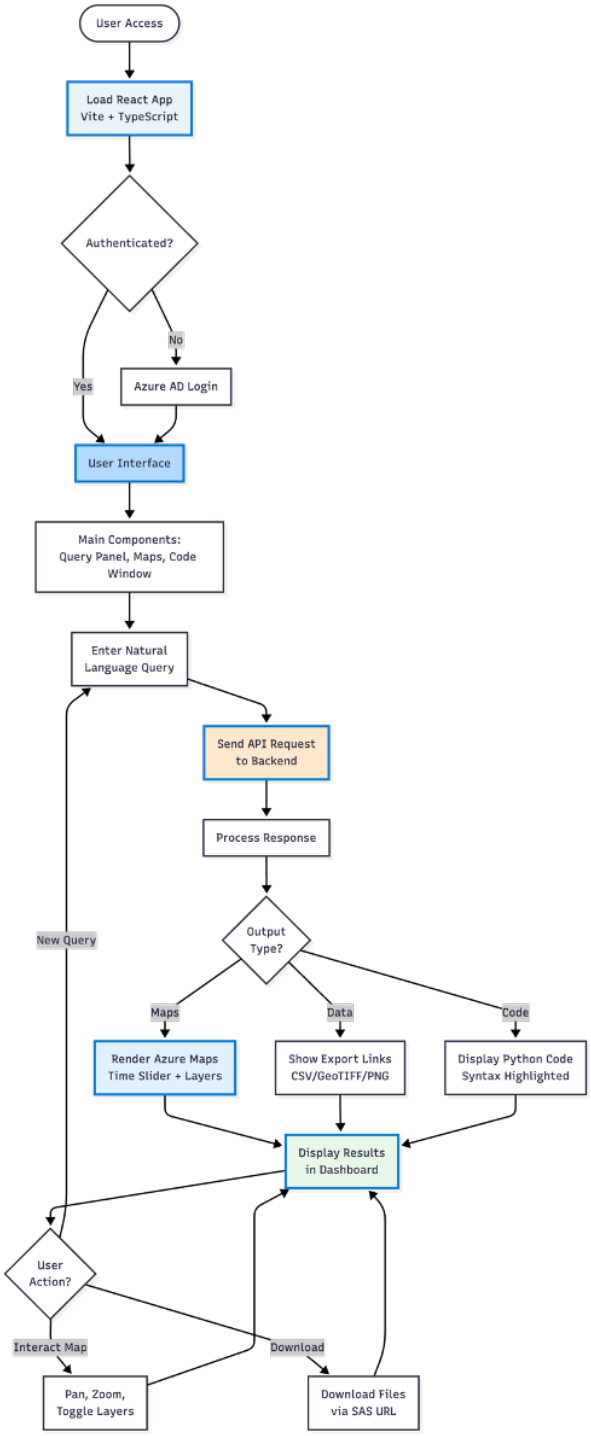
Each query issues a POST request containing the prompt and the locally stored user_id. The backend responds with a structured JSON object that may include static_url, tile_config,

overlay_url, bounds, variable_data, geojson, and map_config. The frontend pads map bounds by 10% for framing while preserving exact georeferencing.

A3.5. Data Export

Users can download static PNG maps with server-managed legends and color scales; client-side rescaling is intentionally disabled to preserve scientific integrity.

Supplementary Figure A2. Frontend user interaction workflow. *Users access the React application (Vite + TypeScript) through Azure AD authentication. The interface provides three output pathways: Maps (Azure Maps with time sliders and layer toggles), Data (CSV/GeoTIFF/PNG exports via SAS URLs), and Code (syntax-highlighted Python scripts). Users can pan, zoom, toggle layers, or download files for local analysis.*



Supplementary Material A4: Chunk Configuration Benchmark

A4.1. Methods

We benchmarked chunk shape against query latency on one month of NLDAS-3 forcing data (31 files, one per day, each containing 24 hourly timesteps; precipitation variable). Five chunk configurations were tested, with dimensions specified as (time, latitude, longitude) in (timesteps, gridpoints, gridpoints): (6, 500, 900), (12, 500, 900), (6, 1000, 1800), (12, 1000, 1800), and (18, 1000, 1800). Each configuration was evaluated under three representative query types of differing complexity: single-point monthly time-series extraction, spatial-mean aggregation over the month, and multi-point time-series extraction. Tests were run with both 4-worker and 8-worker parallelism. We report mean execution time per query type, the overall average across query types, and the per-configuration success rate (fraction of queries completing without error or memory failure).

A4.2. Results

Table A4-1. Chunk-configuration benchmark on NLDAS-3 forcing data (one month, precipitation variable). Times are mean execution time per query type, in seconds. The selected production configuration (6, 500, 900) is shown in bold.

(a) 4-worker setting:

Configuration (time, lat, lon)	Time-series (s)	Spatial mean (s)	Multi-point (s)	Overall avg (s)	Success
(6, 500, 900)	3.14	57.94	0.83	20.6	100%
(18, 1000, 1800)	4.54	54.81	2.32	20.6	100%
(12, 1000, 1800)	2.39	380.04	1.02	127.8	100%
(12, 500, 900)	2.87	739.62	1.05	247.9	100%
(6, 1000, 1800)	15.78	7099.34[1]	34.64	—	78%[2]

(b) 8-worker setting:

Configuration (time, lat, lon)	Time-series (s)	Spatial mean (s)	Multi-point (s)	Overall avg (s)	Success
(12, 1000, 1800)	2.12	30.17	0.77	11.0	100%
(18, 1000, 1800)	6.30	30.87	0.72	12.6	100%
(6, 500, 900)	2.25	36.31	0.83	13.1	100%
(12, 500, 900)	2.08	42.26	0.84	15.1	100%
(6, 1000, 1800)	7.37	195.71	22.25	75.1	100%

[1] Partial failure on the spatial-mean operation (33% success rate at this configuration). [2] Failure attributable to memory pressure on the larger spatial chunk at the 6-timestep temporal block; the operation could not complete within available worker memory.

A4.3. Interpretation and Selection Rationale

The (6, 500, 900) configuration achieved 100% query completion across all three query types at both worker counts, with sub-second multi-point latency and time-series latency below 3.5 seconds. Larger spatial chunks paired with the smaller temporal block, (6, 1000, 1800), exhibited memory pressure and partial failure on the spatial-mean aggregation at 4 workers, indicating that simply increasing spatial chunk size is not a free improvement.

The optimum is query-dependent. Spatial-mean aggregation over the month, which integrates a long temporal extent, is slower under a 6-timestep chunk than under (18, 1000, 1800) (54.81 s vs. 57.94 s at 4 workers, and 30.87 s vs. 36.31 s at 8 workers); batch aggregation workloads therefore benefit from a larger-temporal configuration. We selected (6, 500, 900) for the deployed interactive system because targeted point and multi-point reads, which it serves with the lowest latency and full reliability, are the dominant access pattern of the *Hydrology Copilot*.

We note two limitations of this benchmark. First, the configurations tested are not a full factorial of temporal and spatial dimensions; the 18-timestep temporal block was paired only with the larger 1000×1800 spatial tile, on the basis that smaller spatial tiles would not be expected to scale

advantageously at the larger temporal block given the memory pressure already observed at the (6, 1000, 1800) corner. Second, the benchmark used a single forcing variable (precipitation); multivariable batch computations such as composite drought indices may exhibit different I/O profiles and are not directly characterized by these results. Future work will extend this evaluation to multivariable workloads as they become representative of production access patterns.

Supplementary Material A5: Query Test-Case Suite (Dataset S1)

Referenced by main-text Section 4.3 and the responses to Reviewer 1 Comment 4 and Reviewer 2 Specific Comment 10. The complete spreadsheet, including all columns and the per-case outcome, is attached as Dataset S1. Table A5 below lists the suite; the Result column is to be filled from the executed runs.

We assembled a structured suite of 103 test cases spanning 13 categories (Map 40, Methodology 10, Animation 8, Drought Assessment 8, Edge Case 8, Time Series 6, Comparison 6, Flash Drought 4, Weather Extremes 4, Trend 3, Hydrological Drought 3, Compound Events 2, Crop Impact 1), three complexity levels (Simple 60, Moderate 21, Complex 22), and four data sources (Open Loop 40, Forcing 17, SPI 11, SPI + Open Loop 15, Crop Yield 7, and not-applicable 12 for methodology and edge cases). Each case specifies the exact query, the expected output, and explicit verification criteria. Outputs were verified against the expected reference results rather than by line-by-line code comparison; the system produced correct outputs in all validated cases, with initial code errors caught and corrected by the repair and recovery path (A2.4), and the eight edge cases handled by clarification or graceful rejection rather than execution.

Supplementary Material A6: Benchmark Query Outputs

This section presents representative outputs generated autonomously by the Hydrology Copilot. Each figure reproduces the complete interaction as the user sees it: the natural-language query, the system's narrative summary, the measured end-to-end response time, and the generated visualization. Figures A3.a to A3.f correspond to benchmark queries of Table 4 in the main text, presented in order of increasing complexity tier from basic data access (Tier 1) through advanced statistical workflows (Tier 4); each result was verified against an independently written reference implementation (Section 5.2.1). Response times scale with data volume rather than with analytical complexity. Figure A3.g demonstrates cross-variable statistical analysis beyond the benchmark suite. All reported statistics are computed from the NLDAS-3 data by the executed code rather than generated as text by the language model.

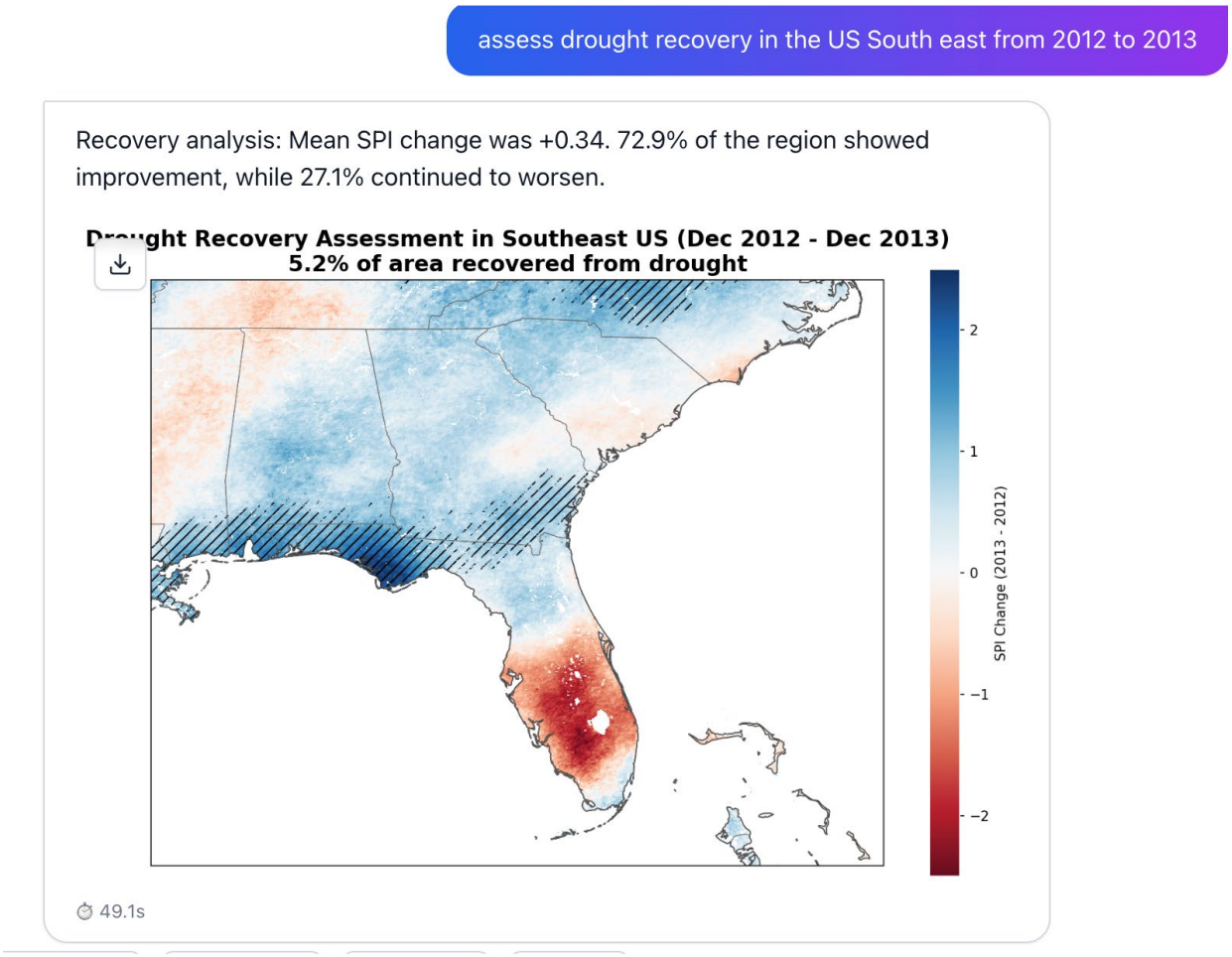
Supplementary Figure A3.a. *Tier 1, basic data access (benchmark Q1): "Show precipitation in Florida on October 12, 2023." The system generates an interactive precipitation map (mm) over the Florida domain with basemap context and a narrative summary of the visualization. Response time: 44.0 s. the agent accumulates hourly precipitation data for this query.*

Supplementary Figure A3.b. *Tier 2, drought indicator (benchmark Q3): "Show the 1-month SPI map for Texas in October 2023." Interactive SPI-1 map over the Texas domain with basemap context and click-for-details interaction, together with a narrative summary computed from the data: mean SPI 0.43 (near normal), moderate drought (SPI ≤ −1.0) over 3.8 percent of the region, severe drought (SPI ≤ −1.5) over 1.1 percent, no extreme drought, and SPI values ranging from −2.18 to 3.04. Response time: 29.8 s.*

Supplementary Figure A3.c. *Tier 2, drought indicator (benchmark Q4): " Identify 10 areas in the Midwest with severe drought in August 2023" Static map highlighting grid cells with SPI-3 values of −1.5 or lower, indicating severe drought. The narrative also states the valid temporal coverage of the Open Loop variables and suggests alternatives, illustrating the guardrail behavior of Section 4.3. Response time: 37.6 s.*

Supplementary Figure A3.d. *Tier 3, flash-drought detection (benchmark Q5): "Detect flash drought in Great Plains: areas where SPI dropped ≥ 2.0 to ≤ −1.5 within 3 months from June to August 2012." SPI map for August 2012 over the Great Plains domain, with hatching marking grid*

cells that meet the flash-drought criterion, an SPI decline of at least 2.0 from June to a value of −1.5 or lower by August, affecting 6.87 percent of the region. The reported percentage is computed from the executed analysis over the valid land cells. Response time: 32.7 s.



Supplementary Figure A3.e. *Tier 3, drought-recovery assessment (benchmark Q6): "Assess drought recovery in the Southeast from 2012 to 2013." SPI change map (December 2012 to December 2013) with hatching over recovered areas, together with summary statistics: mean SPI change +0.34, with 72.9 percent of the region showing improvement. Response time: 49.1 s.*

Supplementary Figure A3.f. *Tier 4, long-term trend analysis (benchmark Q7): "Show the SPI-3 drying trend for the Colorado River Basin from 2001 to 2023 with a trend line." Annual-average SPI-3 series with fitted trend line (slope 0.003 SPI units per year, r = 0.044, p = 0.844). The system correctly reports the trend as not statistically significant rather than overstating a drying signal. Response time: 107 s, dominated by data loading over the multi-year record. The agent reads 274 monthly SPI files and calculates the trend in 107 seconds.*

Supplementary Figure A3.g. *Cross-variable correlation: "Show the correlation between soil moisture and SPI-3 across Texas from 2011 to 2020." The system loaded 120 months of monthly data from two independent sources, Open Loop root-zone soil moisture and the SPI-3 precipitation index, extracted regional means for each month, computed the Pearson correlation ($r = 0.796$, $r^2 = 63.4$ percent), and generated the scatter plot with regression line, demonstrating multi-year, cross-variable statistical analysis beyond single-variable visualization. The response time is 230 seconds.*