# TOWARDS A VIRTUAL NEUROSCIENTIST: AUTONOMOUS NEUROIMAGING ANALYSIS VIA MULTI-AGENT COLLABORATION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Transforming neuroimaging data into clinically actionable biomarkers is a knowledge-intensive and labor-intensive process. Standardized workflows such as fMRIPrep have improved robustness and efficiency, but they are statically configured and cannot reason about downstream objectives, deliberate over alternative strategies, or close the loop between intermediate evidence and subsequent decisions in the way a human researcher would. This lack of closed-loop adaptation often leaves domain experts trapped in a cycle of manual trial-and-error to tune parameters and remediate pipeline failures, severely constraining the scalability of clinical biomarker development. To bridge this gap, we introduce NEXUS, an autonomous multi-agent framework that integrates neuroimaging workflow execution with scientific-objective understanding. Unlike conventional flat tool-calling agents, NEXUS adopts a code-centric execution paradigm where specialist agents collaboratively synthesize and optimize executable programs over composable domain-specific primitives. This design enables robust, long-horizon workflow construction that adapts dynamically to runtime observations. Furthermore, we propose a hierarchical verification framework for autonomous quality control, integrating cohort-level metric screening with agentic visual inspection to drive evidence-grounded workflow remediation. Experiments on ADHD-200 and ADNI demonstrate that NEXUS outperforms automated workflow-based baselines in predictive performance while exhibiting sophisticated agentic behaviors, including strategy exploration and adaptive refinement.

## 1 INTRODUCTION

Neuroimaging analysis plays a central role in modern neuroscience and clinical research, enabling researchers to characterize brain structure and function from modalities such as structural MRI (sMRI) and functional MRI (fMRI). Yet in practice, turning raw neuroimaging data into reliable scientific outcomes remains a knowledge- and labor-intensive process, often with limited efficiency and reproducibility. To alleviate this burden, the neuroimaging community has made major progress toward standardization and automation through data conventions such as BIDS (Gorgolewski et al., 2016) and widely adopted frameworks like fMRIPrep (Esteban et al., 2019), which packages established practices into reusable workflows and has substantially improved robustness and reproducibility. However, these systems are still primarily designed as predefined workflows with bounded configurability: while they can implement well-engineered defaults and limited workflow-internal adaptation, they do not autonomously reason about a downstream scientific objective, explore alternative analytical strategies, or close the loop between intermediate evidence and subsequent decisions in the way a human researcher would during an end-to-end neuroimaging project.

Recent advances in large language models (LLMs) (Naveed et al., 2025) and agentic systems (Gridach et al., 2025) have shown remarkable potential for automating complex knowledge work and scientific discovery, raising a natural question: *Can we build a virtual neuroscientist?* For neuroimaging analysis, however, realizing this vision faces several key challenges. *Ecosystem heterogeneity*: neuroimaging analysis spans diverse software backends with backend-specific interfaces, assumptions, and intermediate representations, while many critical operations are highly specialized and not reliably recoverable from LLM knowledge at runtime. *Long-horizon workflow complexity*:



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

end-to-end analysis requires tightly coupled dataflow, structured control logic, parallel execution, and robust error handling, making conventional flat tool-calling brittle and unreliable. *Quality-control-driven decision making*: neuroimaging quality control is not merely a reporting step, but a judgment-intensive decision point that must be tightly integrated into the workflow to support downstream decisions. *Distributed expertise requirement*: the process involves distinct forms of expertise, including data understanding, image processing, quality assessment, and statistical modeling, which can overload a single monolithic agent and motivate a multi-agent design with specialized roles.

In this work, we take a step toward this goal by introducing NEXUS (Neuroimaging Execution and Understanding System), an autonomous multi-agent framework that bridges neuroimaging execution and scientific understanding. NEXUS addresses the above challenges through four key designs. First, to handle *ecosystem heterogeneity*, we encapsulate specialized neuroimaging routines into composable domain-specific primitives, providing agents with reliable abstractions over diverse backends rather than requiring them to recover low-level domain operations from LLM knowledge. Second, to support *long-horizon workflow construction*, we adopt a code-centric execution paradigm in which specialist agents synthesize executable programs over these primitives, enabling explicit dataflow, structured control logic, parallel execution, and robust error handling beyond flat tool-calling. Third, to realize *quality-control-driven decision making*, we propose an agent-as-a-judge framework for closed-loop autonomous QC, integrating cohort-level metric screening with agentic visual inspection to generate actionable verdicts for dynamic workflow adaptation. Fourth, to meet the *distributed expertise requirement*, NEXUS follows a centralized-planning, decentralized-execution design: a Supervisor Agent maintains global objective coherence, while domain-specific Professional Agents handle data understanding, preprocessing, quality control, and downstream analysis within focused action spaces. We further introduce just-in-time context injection, which dynamically selects task-relevant primitives for each specialist to maintain context efficiency and reduce distraction from irrelevant tools. Instead of assuming that a single fixed workflow should be applied uniformly, we formulate neuroimaging analysis as an adaptive sequential decision-making problem. Starting from raw data and a high-level scientific objective, the agent must orchestrate, refine, and ultimately deliver a workflow aligned with both the data and the end task.

We evaluate NEXUS on two widely used neuroimaging datasets, ADHD-200 (consortium, 2012) and ADNI (Petersen et al., 2010), covering both functional and structural MRI analysis with distinct downstream objectives. The results show that NEXUS outperforms automated workflow-based baselines in predictive performance while maintaining a robust completion rate and exhibiting clear agentic behaviors, such as inter-agent coordination, strategy exploration, and adaptive refinement. Ablation studies show that the architectural components are all necessary for stable long-horizon execution. Finally, we demonstrate that the proposed closed-loop QC module contributes to overall analytical outcomes and achieves moderate agreement with human QC assessments.

## 2 Related Work

**Neuroimaging Workflow Automation.** The neuroimaging community has made substantial progress toward standardization and automation through data standards such as BIDS (Gorgolewski et al., 2016) and widely used software ecosystems, including fMRIPrep (Esteban et al., 2019), FreeSurfer (Fischl, 2012), BrainSuite (Shattuck & Leahy, 2002), Nipype (Gorgolewski et al., 2011), and MRIQC (Esteban et al., 2017). These tools have improved robustness, reproducibility, and usability by packaging standard practices into reusable workflows. However, they largely rely on statically predefined processing pipelines, offer limited configurability, and treat QC as a post hoc reporting mechanism rather than a decision-making component, preventing them from reasoning about downstream scientific objectives, comparing alternative analytical strategies, or adapting execution based on intermediate evidence. Our work advances beyond this line of research by enabling autonomous, objective-aware, and feedback-driven analysis.

**LLM Agents for Scientific Workflows.** Recent work has increasingly explored LLM agents not only for general tool use, but also for scientific discovery and domain-specialized research automation. For example, ReAct (Yao et al., 2023) established a general reasoning-and-acting paradigm, while subsequent systems explored multi-agent collaboration and executable-code-based action spaces such as CodeAct (Wang et al., 2024). In parallel, scientific agent systems such as The AI Scientist (Lu et al., 2024) aim to automate substantial parts of the research cycle, and domain-specific agent such as Biomni (Huang et al., 2025) demonstrates that coupling LLMs with expert tools, databases, and executable environments can expand scientific task coverage. During the writing of

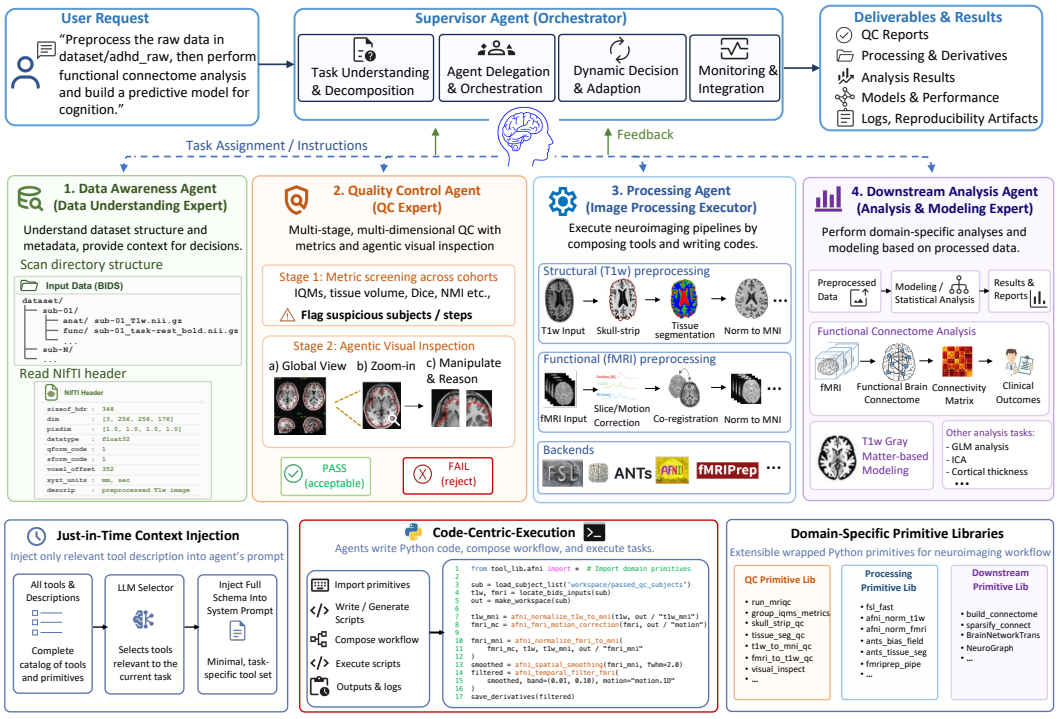


Figure 1: Overview of the NEXUS framework.

this work, we noticed several recent works, including NeuroClaw (Wang et al., 2026), NEURA (Xie et al., 2026), Agentic LLMs (Erdur et al., 2026), NeuroAgent (Zhong et al., 2026), NeuroPilot (Chen et al., 2026), and BrainPilot (Li et al., 2026), which share a similar spirit with us in exploring agentic AI for neuroimaging research. They primarily focus on task completion through tool orchestration, reproducible execution, and traceable workflows, typically instantiating predefined or skill-encapsulated pipelines. In contrast, NEXUS targets fully autonomous, end-to-end neuroimaging analysis, emphasizing adaptive workflow construction, dynamic refinement, objective-aware optimization, and closed-loop agentic QC.

# 3 METHOD

## 3.1 PROBLEM FORMULATION AND SYSTEM ARCHITECTURE

**Problem Formulation.** We formalize autonomous neuroimaging analysis as a goal-oriented sequential decision-making process. Let $\mathcal{D}$ denote a raw neuroimaging dataset (e.g., T1w and fMRI in BIDS format) and $\mathcal{G}$ a high-level research objective expressed in natural language. The goal is to identify an optimal execution trace $\mathcal{T} = \{(a_1, o_1), (a_2, o_2), \ldots, (a_T, o_T)\}$ that transforms $\mathcal{D}$ into a set of deliverables $\mathcal{R}$ (e.g., analytical results and validated derivatives). Here, $a_t$ denotes an action (e.g., executing a processing primitive or performing QC), and $o_t$ the observation at time $t$, capturing the updated state and execution feedback. Unlike predefined pipelines, this process is inherently adaptive: each subsequent action $a_{t+1}$ is conditioned on the cumulative history of reasoning and environmental feedback, i.e., $a_{t+1} = \pi(\mathcal{G}, \mathcal{D}, \mathcal{T}_{1:t})$, with $\pi$ denoting the agentic policy.

**Architecture.** We instantiate this formulation using a hierarchical multi-agent architecture with centralized planning and decentralized execution. As illustrated in Figure 1, the system consists of a high-level Supervisor Agent and a cohort of Domain-Specific Professional Agents: **Supervisor Agent** serves as the central orchestrator. It translates the objective $\mathcal{G}$ into an executable workflow, dispatches tasks, coordinates inter-agent collaboration, and performs dynamic decision-making based on runtime feedback; **Data Awareness Agent** extracts task-related dataset context (e.g., directory structure or metadata) to ground supervisor's planning decisions; **Quality Control Agent** operates as the critic, performing hierarchical quality assessment of neuroimaging data and issuing evidence-backed verdicts for closed-loop workflow remediation and adaptation; **Processing Agent**

instantiates and executes processing plans by composing primitives (e.g., skull stripping, registration) into runnable scripts, producing derivatives with provenance; **Downstream Analysis Agent** conducts statistical analysis or predictive modeling (e.g., connectome analysis) on QC-validated derivatives, translating processed data into objective-aligned analytical outcomes.

This architecture is designed to decouple strategic global reasoning from specialized local execution. Centralized planning ensures global objective coherence and consistent decision-making across the long-horizon neuroimaging workflow. Decentralized execution allows each specialist to operate on bounded, domain-specific contexts with well-scoped action spaces, mitigating cognitive overload and reasoning drift typically observed in monolithic agents. This modular nature affords an evolvable ecosystem where new capabilities can be added through simple expert registration.

### 3.2 Code-Centric Execution with Composable Primitives

Neuroimaging analysis with LLM agents presents several intertwined challenges: (i) The neuroimaging ecosystem is highly heterogeneous, comprising numerous software packages and specialized analytical routines, each with distinct interfaces, implicit assumptions, and incompatible intermediate representations. Naively exposing these low-level utilities to an agent exacerbates hallucination risks and unstable execution, and often leads to weak cross-step compositionality. Moreover, many critical steps are highly domain-specific and sparsely represented in LLM training data, making it unreliable for an agent to implement them at runtime. (ii) In conventional tool-calling paradigms, complex workflows are realized as sequences of discrete tool invocations. While sufficient for short interactions, this design becomes fragile for long-horizon scientific workflows that require tightly coupled dataflow, structured control logic (e.g., iterations over cohorts, conditional branching), parallel execution, and robust error handling. As task complexity scales, these limitations manifest as brittle compositionality and reduced execution reliability. To address these challenges, inspired by CodeAct (Wang et al., 2024), we adopt a code-centric execution paradigm built upon composable domain-specific primitives. Instead of exposing raw tools as flat callable actions, we encapsulate heterogeneous functionalities into standardized computational primitives and empower agents to synthesize flexible programs that orchestrate them.

#### 3.2.1 Domain-Specific Primitives

We define a primitive as an atomic, composable operation over neuroimaging artifacts. Formally, each primitive $p \in \mathcal{P}$ is a Python callable, specified by a *card* with (i) a name, module path, and short description, and (ii) a detailed I/O schema describing required inputs, outputs, and side effects, wrapping established software (e.g., neuroimaging processing softwares, QC softwares) or custom utilities, and returning structured outputs:

$$p : (x, \theta) \rightarrow (x', r), \tag{1}$$

where $x$ are input artifacts, $\theta$ are parameters, $x'$ are output artifacts, and $r$ is the returned result (e.g., QC verdicts, predictive performance). Primitives are organized into three families aligned with Professional Agents: (a) Processing primitives implement image transformations and preprocessing steps across sMRI and fMRI; (b) QC primitives compute quantitative metrics and generate visualization elements for inspection; (c) Analysis primitives implement downstream feature construction and modeling components (e.g., ROI time-series extraction, connectome construction, predictive models). These primitives provide a standardized abstraction layer over heterogeneous neuroimaging backends and specialized analytical routines, shielding agents from low-level interface differences and reducing the need to synthesize fragile domain-specific implementations at runtime.

#### 3.2.2 Just-in-Time Context Injection

Naively injecting all primitive descriptions and schemas into the prompt can lead to context inflation and degraded reasoning, particularly with many available primitives. We therefore introduce Just-in-Time (JIT) Context Injection: before a Professional Agent enters its execution loop, it retrieves from the primitive library and injects only the *minimal sufficient* context into its system prompt.

Each Professional Agent owns a primitive library $\mathcal{P}$, where each primitive $p \in \mathcal{P}$ is associated with a name $n_p$ and a short description $d_p$, in addition to a full schema (parameters, preconditions, outputs). We build a compact index consisting only of $\{(n_p, d_p)\}_{p \in \mathcal{P}}$. Given a Supervisor-issued

instruction $I$, a selector produces a task-relevant subset of primitives:

$$\mathcal{P}_I = \texttt{SelectTools}(I, \{(n_p, d_p)\}_{p \in \mathcal{P}}), \tag{2}$$

where $\texttt{SelectTools}(\cdot)$ denotes a single LLM invocation prompted to select a task-relevant subset of primitives from the compact index, and $\mathcal{P}_I \subseteq \mathcal{P}$ (typically $|\mathcal{P}_I| \ll |\mathcal{P}|$). Only primitives in $\mathcal{P}_I$ have their full schemas injected into the agent prompt. This just-in-time injection (i) reduces token load, (ii) mitigates distraction from irrelevant affordances, and (iii) improves modular extensibility since adding new primitives does not linearly inflate every prompt.

#### 3.2.3 CODE-CENTRIC SYNTHESIS

Instead of emitting sequential, isolated tool calls, the core action of the Professional Agents is the synthesis and execution of programs. Given a supervisor-issued instruction $I$ and the professional's own cumulative history $\mathcal{T}$, a professional agent can produce an executable program $\mathcal{C}$ that composes primitives with explicit control flow:

$$\mathcal{C} = \textsc{Synthesize}(I, \mathcal{P}_I, \mathcal{T}), \tag{3}$$

where $\mathcal{P}_I$ is the task-related primitive set selected via just-in-time context injection. The program $\mathcal{C}$ is executed in the runtime environment, and its outputs (artifacts, logs, metrics) are incorporated back into $\mathcal{T}$ for subsequent reasoning. Compared with flat tool invocation, this "code-as-act" design enables expressive control logic, explicit data provenance, and scalable composition, supporting flexible and reproducible long-horizon neuroimaging workflows.

In addition to code synthesis over domain-specific primitives, professional agents are also equipped with a minimal set of generic execution utilities (e.g., file I/O and controlled shell execution) to interface with the runtime environment and manage artifacts.

### 3.3 AGENT-AS-A-JUDGE FOR CLOSED-LOOP AUTONOMOUS QC

A central bottleneck in neuroimaging analysis is that *QC is both indispensable and difficult to automate*. Existing pipelines can generate QC reports, but converting reports into actionable decisions (e.g., exclusion, parameter adjustment, or alternative backends) remains largely manual. We formalize QC as an evidence-grounded judgment problem and instantiate the Quality Control Agent as an *Agent-as-a-Judge* operating in a closed loop with the Supervisor and Processing Agents.

#### 3.3.1 HIERARCHICAL QC AS STRUCTURED JUDGMENT

Let $x$ denote a neuroimaging artifact (raw data or derivative). The QC Agent adaptively constructs a set of quantitative image quality metrics $\phi(x)$ and corresponding visualization artifacts $v(x)$ conditioned on the processing stage and the provenance of $x$. The QC problem is to produce a binary verdict $y \in \{\texttt{PASS}, \texttt{FAIL}\}$ and an evidence bundle $e$ that supports the decision:

$$(y, e) = \mathcal{J}(x, \phi(x), v(x)), \tag{4}$$

where $\mathcal{J}$ is the structured judgment conducted by the QC Agent. The verdict $y$ and evidence $e$ are fed back to the Supervisor, which conditions subsequent planning and processing strategies on QC outcomes, forming a closed-loop adaptive workflow.

Concretely, QC agent adopts a two-stage hierarchical strategy. First, a metric-based screening stage performs cohort-level outlier detection over $\{\phi(x_i)\}_{i=1}^{N}$, identifying a subset $\mathcal{S} \subset \{1, \ldots, N\}$ of potentially problematic subjects, which reduces the search space and enables scalable automation. Second, for $i \in \mathcal{S}$, the agent further performs fine-grained visual inspection to determine the final verdict. This hierarchy mirrors expert practice.

#### 3.3.2 AGENTIC VISUAL INSPECTION

To mimic the specialized "eye" of a neuroscientist, we empower the QC Agent with Agentic Visual Inspection. When a visual check is needed, the agent invokes a Vision-Language Model (VLM) tasked with a multi-turn diagnostic protocol. The visualization $v(x)$ constructed by QC agent is passed to the VLM configured with: (i) explicit scoring criteria describing acceptable and unacceptable patterns; (ii) curated few-shot exemplars illustrating typical success and failure modes; and (iii) access to a controlled programming tool enabling iterative region-of-interest cropping and zoom-in

inspection. Instead of producing a single-shot judgment, the VLM engages in iterative analysis: it may proactively zoom into fine-grained details with the given programming tool, inspect boundary regions, and accumulate intermediate observations before emitting a final verdict. This *program-augmented visual reasoning* mitigates coarse global judgments and improves sensitivity to details.

# 4 EXPERIMENTS

To evaluate the efficacy and autonomy of NEXUS, we design experiments to answer three primary research questions: **(RQ1)** Can the system transform raw neuroimaging data and a high-level scientific objective into well-adapted analytical workflows and outcome-oriented solutions through autonomous exploration, iterative refinement, and dynamic decision-making? **(RQ2)** Are the architectural design choices of the system necessary for stable and efficient long-horizon execution? **(RQ3)** Does the proposed QC module contribute meaningfully to overall analytical outcomes, and to what extent do its judgments align with human assessment?

## 4.1 EXPERIMENTAL SETUP

**Datasets and Tasks.** We evaluate NEXUS on two widely used neuroimaging datasets that represent distinct modalities and downstream objectives: ADHD-200 (consortium, 2012) (fMRI, $N = 200$; 100 ADHD, 100 Healthy Controls) and ADNI (Petersen et al., 2010) (sMRI, $N = 200$; 100 Alzheimer's Disease, 100 Cognitively Normal). For each dataset, the agent is provided with a natural-language objective specifying the prediction target and required deliverables, and is tasked with developing a complete neuroimaging analysis workflow from the raw training data. This requires the agent to inspect the dataset, design and orchestrate an appropriate preprocessing and quality-control strategy, carry out downstream analysis and model training, and adaptively refine the workflow based on intermediate results and execution feedback. The final deliverable consists of the developed preprocessing pipeline, and the trained model artifacts along with the corresponding inference procedure, such that the resulting analysis package can be readily reusable and generalized to unseen held-out subjects for final evaluation. The two datasets instantiate this general setting with different modality-specific objectives. For ADHD-200, the agent derives an end-to-end workflow from raw fMRI data for functional connectome-based diagnosis prediction. For ADNI, the corresponding objective is to derive such a workflow from raw T1-weighted MRI for structural diagnosis prediction based on voxel-wise gray matter features. Detailed tasks are provided in Appendix A.1.

**Baselines.** We compare NEXUS with automated workflow-based neuroimaging baselines spanning different levels of workflow search. The standard baselines combine established preprocessing workflows with automated downstream model selection on the training set. Concretely, the evaluated preprocessing backbones include fMRIPrep (Esteban et al., 2019), AFNI (Cox, 1996), FSL (Jenkinson et al., 2012), SPM12 (Ashburner et al., 2014), and ANTs (Avants et al., 2009). The first three tools support both structural and functional MRI and are thus applied to both datasets. Since SPM12 is primarily designed for fMRI, we evaluate it only on ADHD, whereas ANTs, tailored for sMRI, is evaluated only on ADNI. For these baselines, preprocessing follows standard workflow configurations commonly used for each software package, while downstream analysis is automatically optimized through cross-validation over model classes, hyperparameters, and task-specific settings on the training set. After model selection, the chosen model is retrained on the full training set and evaluated once on the held-out test set. We further include an *Exhaustive Workflow Search* baseline that extends automated search to the full analysis workflow by systematically exploring the joint space of preprocessing pipelines, QC choices, and downstream modeling configurations. Its search space and evaluation budget are intentionally more generous than those explored by NEXUS, providing a complementary comparison in which a broad set of workflow configurations is evaluated exhaustively rather than selected adaptively based on agentic decision-making. Detailed definitions of the candidate configurations and search procedure, together with specifications of the standard baselines, are provided in Appendix A.2.

**Evaluation Protocol.** For each dataset, we use a stratified 4:1 split of the 200 subjects: 160 subjects for workflow development, cross-validation, and model selection, and 40 subjects held out for final evaluation. Throughout workflow development and execution, NEXUS has access only to the training set; the held-out test set remains inaccessible to the system and is used only for final evaluation. NEXUS and all baselines are run five times; we report mean held-out AUC, F1, and Accuracy. For NEXUS, we also report execution statistics. Agents use `gpt-5.2` with default settings, and visual QC uses `gemini-3-flash-preview`.

Table 1: End-to-end downstream prediction performance (%) on held-out test sets. DS denotes the automated downstream analysis used on top of each predefined preprocessing backbone.

| | ADHD | | | | ADNI | | |
|---|---|---|---|---|---|---|---|
| **Method** | **AUC** | **F1** | **ACC** | **Method** | **AUC** | **F1** | **ACC** |
| fMRIPrep + DS | $61.30 \pm 2.75$ | $61.89 \pm 6.32$ | $54.50 \pm 3.67$ | fMRIPrep + DS | $90.60 \pm 1.62$ | $81.06 \pm 2.40$ | $83.00 \pm 1.87$ |
| AFNI + DS | $62.60 \pm 5.76$ | $58.09 \pm 6.48$ | $57.00 \pm 3.67$ | AFNI + DS | $89.30 \pm 0.46$ | $79.56 \pm 0.89$ | $82.00 \pm 1.00$ |
| FSL + DS | $55.40 \pm 6.50$ | $61.40 \pm 4.80$ | $55.00 \pm 5.20$ | FSL + DS | $80.90 \pm 0.90$ | $69.50 \pm 3.90$ | $71.00 \pm 3.70$ |
| SPM + DS | $57.20 \pm 10.30$ | $64.50 \pm 3.40$ | $54.00 \pm 7.40$ | ANTs + DS | $89.30 \pm 0.50$ | $75.10 \pm 3.90$ | $76.50 \pm 3.40$ |
| Exhaustive Search | $57.45 \pm 3.70$ | $49.85 \pm 7.25$ | $50.50 \pm 2.92$ | Exhaustive Search | $92.55 \pm 3.11$ | $81.02 \pm 3.12$ | $82.50 \pm 3.87$ |
| **NEXUS (Ours)** | $\mathbf{66.45 \pm 4.03}$ | $\mathbf{65.70 \pm 3.01}$ | $\mathbf{64.50 \pm 3.71}$ | **NEXUS (Ours)** | $\mathbf{93.00 \pm 1.94}$ | $\mathbf{82.83 \pm 2.46}$ | $\mathbf{84.50 \pm 2.09}$ |

### 4.2 End-to-End Autonomous Neuroimaging Analysis (RQ1)

**Downstream Predictive Performance.** Table 1 compares the held-out test performance of NEXUS against automated workflow-based baselines. Across both datasets, NEXUS achieves the best overall performance. It also outperforms the Exhaustive Workflow Search baseline, despite the latter systematically evaluating a larger space of preprocessing, QC, and downstream modeling configurations. These results suggest that broad workflow enumeration alone does not necessarily translate into better final performance. The exhaustive or predefined search procedures are automated but static, whereas NEXUS dynamically develops and adapts its analysis strategy based on dataset characteristics, task requirements, QC outcomes, and intermediate execution feedback. To further disentangle the contribution of agentic decision-making across different stages of the workflow, we perform component-wise analyses over preprocessing pipeline construction, quality control, and downstream optimization; detailed results are provided in Appendix A.3.

Table 2: Summary statistics across five independent runs of NEXUS. Values are reported as mean (range) across runs. Completion denotes producing all required deliverables; adaptive modeling refinement denotes downstream-analysis iterations triggered by runtime observations such as validation performance, feature instability, or overfitting risk.

| **Statistic** | **ADHD** | **ADNI** |
|---|---|---|
| **Global Execution Footprint** | | |
| Completion Rate | 5/5 | 5/5 |
| Runtime | 54.4 h (27.8–70.9) | 38.0 h (23.7–83.2) |
| API Cost | \$9.01 (6.77–11.94) | \$6.46 (3.61–10.68) |
| # inter-agent interactions | 25 (18–32) | 26 (20–30) |
| # agent actions | 158 (132–200) | 126 (82–159) |
| # executable Python scripts | 42 (27–64) | 29 (16–48) |
| **Exploration and Adaptation** | | |
| # candidate preprocessing pipelines explored | 2 (1–3) | 2 (2–3) |
| # adaptive modeling refinement rounds | 3 (2–5) | 2 (1–4) |
| # subjects passing QC | 130 (126–135) | 148 (140–154) |
| QC checks conducted | Raw T1w/BOLD MRIQC; fMRI→T1w coregistration; fMRI→MNI QC | Raw T1w MRIQC; skull-stripping; tissue-segmentation; T1w→MNI QC |
| Delivered preprocessing pipeline | AFNI-based (2/5), fMRIPrep-based (3/5) | ANTs-based (3/5), ANTs+FSL Hybrid (2/5) |
| Delivered downstream modeling | Site-aware connectome classification; AAL90/Schaefer200/HCP360 + Pearson/Fisher-z + regularized LR or BrainNetTransformer | Voxel-wise VBM classification; univariate feature selection + LR/ElasticNet |

**Agentic Dynamics and Execution Statistics.** Table 2 shows that NEXUS is not merely executing a fixed automated recipe, but actively developing and refining workflows through multi-agent coordination and runtime adaptation. Across both datasets, it achieves a 100% completion rate

over all independent runs, while exploring multiple candidate preprocessing pipelines, conducting stage-specific QC, and performing several rounds of downstream refinement before issuing a final deliverable. This pattern indicates a context-aware planning and dynamic adjustment process rather than one-shot execution, which explains why NEXUS consistently outperforms automated workflow-based baselines. The delivered workflows vary across datasets and runs—e.g., AFNI- versus fMRIPrep-based preprocessing with different connectome modeling configurations on ADHD-200, and pure ANTs versus ANTs+FSL hybrid preprocessing with different voxel-wise VBM modeling strategies on ADNI—showing that the system does not commit to a single hard-coded software stack or downstream recipe, but instead makes data- and task-dependent decisions. The substantial number of supervisor-subagent interactions and executable scripts further shows that successful execution relies on sustained multi-agent coordination and code-centric orchestration, which naturally accommodates the complex dataflows and dependencies inherent in neuroimaging analysis. Detailed running trajectory can be found in Appendix A.5.

### 4.3 Architectural Ablations (RQ2)

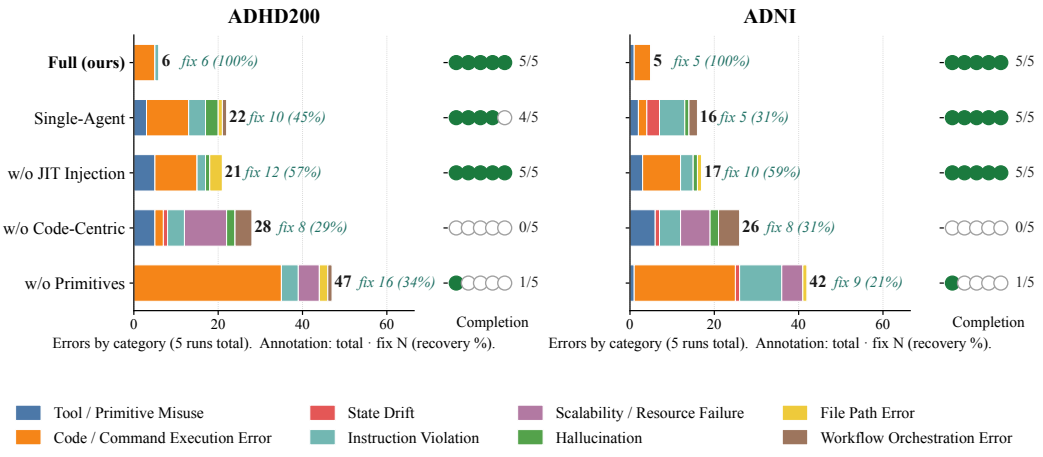


Figure 2: Ablation study results. Stacked bars show total execution errors across five independent runs. Annotations report total error count, number of autonomously recovered errors, and recovery rate. Filled and open circles indicate completed and failed runs.

We next evaluate whether the main architectural designs of NEXUS are necessary for stable execution. We run the ablation study on subsampled 20-subject subsets of ADHD-200 and ADNI. The task formulation remains the same as in the full end-to-end setting, but the goal here is not to compare downstream predictive performance. Instead, we focus on execution stability, completion and failure characteristics. We compare the following variants: (1) **Single-Agent**, which collapses all roles into a single monolithic agent handling the whole workflow of neuroimaging analysis; (2) **w/o JIT Context Injection**, which removes task-adaptive primitive selection and injects the full primitive library into each sub-agent prompt; (3) **w/o Code-Centric Execution**, which replaces executable program synthesis with direct, flat tool calling; and (4) **w/o Primitive Abstraction**, which removes the encapsulated primitives and requires agents to rely on low-level commands or self-written implementation logic. All variants are evaluated under the same execution budget, with a maximum of 500 ReAct recursion (observe $\rightarrow$ act $\rightarrow$ ...) limits and 12 hours of runtime.

As illustrated in Figure 2, the full system NEXUS consistently achieves a 100% completion rate with fewest execution errors and uniquely maintains a 100% autonomous error recovery rate. In contrast, degrading the architecture exposes distinct failure modes: Both the Single-Agent and w/o JIT Injection variants highlight the vulnerability of LLMs to cognitive and contextual overload. Forcing a monolithic agent to handle both global planning and local execution overwhelms its reasoning capacity, leading to State Drift, Hallucination, and sharply degraded error recovery. Similarly, omitting JIT injection bloats the prompt with irrelevant tool schemas, directly inducing elevated Tool/Primitive Misuse and Instruction Violations. The w/o Code-Centric variant fails catastrophically (0/5 completion on both datasets). It suffers from severe Scalability/Resource Failure and Workflow Orchestration Errors, demonstrating the necessity of code-centric paradigm for

long-horizon scientific workflows that require robust control flow and efficient parallel execution. Removing primitive abstraction causes one of the most severe performance drops. Without pre-encapsulated domain-specific operations, the agents must directly generate low-level commands and implementation details for a highly heterogeneous and specialized neuroimaging ecosystem. This substantially increases command misuse, artifact mismatch, and malformed execution scripts. Detailed ablation records and error taxonomy can be found in Appendix B.

### 4.4 Evaluation of Closed-Loop Autonomous Quality Control (RQ3)

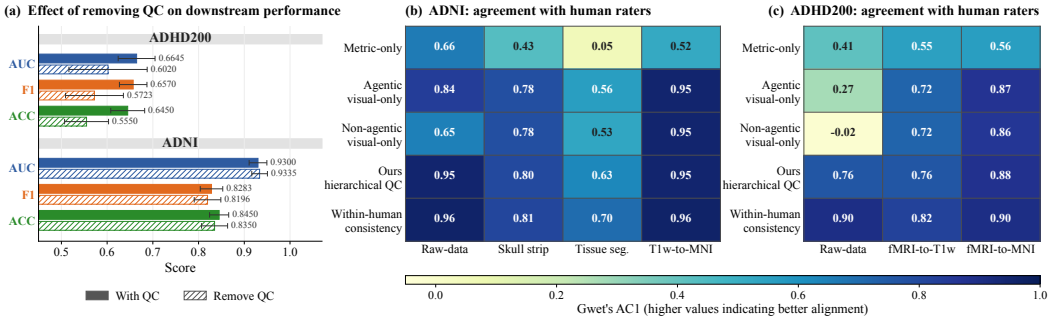


Figure 3: Evaluation of the closed loop autonomous QC module.

**Impact of QC Filtering on Downstream Performance.** As shown in Figure 3(a), removing the QC filtering stage (i.e., keeping all subjects into downstream modeling) leads to a significant performance degradation on the ADHD-200 dataset. On the ADNI dataset, we observe a more modest decrease in F1 and Accuracy. This divergence is consistent with the underlying data characteristics: ADHD-200 consists of noisier fMRI data, making downstream performance more sensitive to quality variation, whereas ADNI consists of generally higher-quality sMRI data, with fewer subjects excluded by QC. Overall, these results demonstrate that the proposed closed-loop QC effectively intercepts artifacts and poor-quality features, preventing them from confounding downstream analysis and improving the robustness of the final scientific deliverables.

**Alignment with Human Assessments.** Figure 3(b) and (c) reports the agreement between each QC variant and human raters across multiple checkpoints, measured by Gwet's AC1 (Gwet, 2008), an inter-rater agreement coefficient suitable for highly imbalanced settings, with higher values indicating better alignment. Each reported value is averaged over comparisons to the three human raters. The proposed hierarchical QC achieves the strongest agreement with human judgments across all checkpoints, indicating that metric-based screening followed by visual inspection better captures both quantitative abnormalities and qualitative anatomical cues. Agentic visual QC also generally aligns better with human raters than non-agentic visual QC, indicating the value of iterative visual exploration for specialized neuroimaging judgment. Detailed QC evaluation settings and human assessments can be found in Appendix C.

## 5 Conclusion

In this work, we introduced NEXUS, a novel multi-agent system that advances neuroimaging analysis from statically predefined pipelines to autonomous, adaptive, and closed-loop scientific workflows. To overcome the high heterogeneity and long-horizon complexity inherent in neuroimaging, NEXUS leverages a code-centric execution paradigm over composable domain-specific primitives, effectively grounded by just-in-time context injection. Furthermore, we addressed the critical bottleneck of manual quality control by proposing a hierarchical, agent-as-a-judge framework. By integrating cohort-level quantitative metric screening with program-augmented agentic visual inspection, NEXUS successfully translates quality assessment into evidence-driven workflow remediation. Experiments on ADHD-200 and ADNI show that NEXUS outperforms automated workflow-based baselines, exhibits clear agentic dynamics and relies critically on its architectural components for stable long-horizon execution. We further show that closed-loop QC both contributes to downstream outcomes and aligns well with human assessment. More broadly, these results highlight the promise of agentic neuroimaging analysis. Given its modular and extensible design, NEXUS provides a foundation for future systems supporting broader neuroimaging modalities, richer primitive libraries, and more diverse downstream tasks.

## REFERENCES

John Ashburner, Gareth Barnes, Chun-Chuan Chen, Jean Daunizeau, Guillaume Flandin, Karl Friston, Stefan Kiebel, James Kilner, Vladimir Litvak, Rosalyn Moran, et al. Spm12 manual. *Wellcome Trust Centre for Neuroimaging, London, UK*, 2464(4):53, 2014.

Brian B Avants, Charles L Epstein, Murray Grossman, and James C Gee. Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain. *Medical Image Analysis*, 12(1):26–41, 2008.

Brian B Avants, Nick Tustison, Gang Song, et al. Advanced normalization tools (ants). *Insight j*, 2 (365):1–35, 2009.

Brian B Avants, Nicholas J Tustison, Gang Song, Philip A Cook, Arno Klein, and James C Gee. A reproducible evaluation of ANTs similarity metric performance in brain image registration. *NeuroImage*, 54(3):2033–2044, 2011a.

Brian B Avants, Nicholas J Tustison, Jue Wu, Philip A Cook, and James C Gee. An open source multivariate framework for n-tissue segmentation with evaluation on public data. *Neuroinformatics*, 9(4):381–400, 2011b.

Yiyao Chen, Yucheng Li, Jungong Tong, Shaoqi Wang, Kunhao Zhou, Ziquan Wei, Monica Murea, Marissa DiPiero, Tingting Dan, and Guorong Wu. Neuropilot: An agent-driven smart pipeline for processing, quality control, and managing neuroimages. *arXiv preprint arXiv:2608.07541*, 2026.

ADHD-200 consortium. The adhd-200 consortium: a model to advance the translational potential of neuroimaging in clinical neuroscience. *Frontiers in systems neuroscience*, 6:62, 2012.

Robert W Cox. Afni: software for analysis and visualization of functional magnetic resonance neuroimages. *Computers and Biomedical research*, 29(3):162–173, 1996.

Ayhan Can Erdur, Daniel Scholz, Jiazhen Pan, Benedikt Wiestler, Daniel Rueckert, and Jan C Peeken. Agentic large language models for training-free neuro-radiological image analysis. *arXiv preprint arXiv:2604.16729*, 2026.

Oscar Esteban, Daniel Birman, Marie Schaer, Oluwasanmi O Koyejo, Russell A Poldrack, and Krzysztof J Gorgolewski. Mriqc: Advancing the automatic prediction of image quality in mri from unseen sites. *PloS one*, 12(9):e0184661, 2017.

Oscar Esteban, Christopher J Markiewicz, Ross W Blair, Craig A Moodie, A Ilkay Isik, Asier Erramuzpe, James D Kent, Mathias Goncalves, Elizabeth DuPre, Madeleine Snyder, et al. fmriprep: a robust preprocessing pipeline for functional mri. *Nature methods*, 16(1):111–116, 2019.

Bruce Fischl. Freesurfer. *Neuroimage*, 62(2):774–781, 2012.

Matthew F Glasser, Timothy S Coalson, Emma C Robinson, Carl Hacker, John Harwell, Essa Yacoub, Kamil Ugurbil, Jesper Andersson, Christian F Beckmann, Mark Jenkinson, et al. A multimodal parcellation of human cerebral cortex. *Nature*, 2016.

Krzysztof Gorgolewski, Christopher D Burns, Cindee Madison, Dav Clark, Yaroslav O Halchenko, Michael L Waskom, and Satrajit S Ghosh. Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Frontiers in neuroinformatics*, 5:13, 2011.

Krzysztof J Gorgolewski, Tibor Auer, Vince D Calhoun, R Cameron Craddock, Samir Das, Eugene P Duff, Guillaume Flandin, Satrajit S Ghosh, Tristan Glatard, Yaroslav O Halchenko, et al. The brain imaging data structure, a format for organizing and describing outputs of neuroimaging experiments. *Scientific data*, 3(1):1–9, 2016.

Mourad Gridach, Jay Nanavati, Khaldoun Zine El Abidine, Lenon Mendes, and Christina Mack. Agentic ai for scientific discovery: A survey of progress, challenges, and future directions. *arXiv preprint arXiv:2503.08979*, 2025.

Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008.

Kexin Huang, Serena Zhang, Hanchen Wang, Yuanhao Qu, Yingzhou Lu, Yusuf Roohani, Ryan Li, Lin Qiu, Gavin Li, Junze Zhang, et al. Biomni: A general-purpose biomedical ai agent. *biorxiv*, 2025.

Mark Jenkinson, Peter Bannister, Michael Brady, and Stephen Smith. Improved optimization for the robust and accurate linear registration and motion correction of brain images. *NeuroImage*, 17(2):825–841, 2002.

Mark Jenkinson, Christian F Beckmann, Timothy E J Behrens, Mark W Woolrich, and Stephen M Smith. FSL. *NeuroImage*, 62(2):782–790, 2012.

Xuan Kan, Wei Dai, Hejie Cui, Zilong Zhang, Ying Guo, and Carl Yang. Brain network transformer. In *NeurIPS*, 2022.

Haoxuan Li, Tianci Gao, Jianhe Li, Yang Fan, Runze Shi, Weiran Wang, Tianxiang Zhao, Zezhao Wu, Xiaoyang Jiang, Qihui Zhang, et al. Brainpilot: Automating brain discovery with agentic research. *arXiv preprint arXiv:2607.15079*, 2026.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–72, 2025.

William D Penny, Karl J Friston, John T Ashburner, Stefan J Kiebel, and Thomas E Nichols. *Statistical Parametric Mapping: The Analysis of Functional Brain Images*. Elsevier, 2011.

Ronald Carl Petersen, Paul S Aisen, Laurel A Beckett, Michael C Donohue, Anthony Collins Gamst, Danielle J Harvey, Clifford R Jack Jr, William J Jagust, Leslie M Shaw, Arthur W Toga, et al. Alzheimer's disease neuroimaging initiative (adni) clinical characterization. *Neurology*, 74(3): 201–209, 2010.

Anwar Said, Roza G. Bayrak, Tyler Derr, Mudassir Shabbir, Daniel Moyer, Catie Chang, and Xenofon Koutsoukos. Neurograph: benchmarks for graph machine learning in brain connectomics. In *NeurIPS*, 2023.

Alexander Schaefer, Ru Kong, Evan Gordon, Timothy Laumann, Xinian Zuo, Avram Holmes, Simon Eickhoff, and T Thomas Yeo. Local-global parcellation of the human cerebral cortex from intrinsic functional connectivity mri. *Cerebral Cortex*, 2017.

David W Shattuck and Richard M Leahy. Brainsuite: an automated cortical surface identification tool. *Medical image analysis*, 6(2):129–142, 2002.

Stephen M Smith. Fast robust automated brain extraction. *Human Brain Mapping*, 17(3):143–155, 2002.

Nicholas J Tustison, Brian B Avants, Philip A Cook, Yuanjie Zheng, Alexander Egan, Paul A Yushkevich, and James C Gee. N4ITK: improved N3 bias correction. *IEEE Transactions on Medical Imaging*, 29(6):1310–1320, 2010.

Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Octave Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *Neuroimage*, 2002.

Cheng Wang, Zhibin He, Zhihao Peng, Shengyuan Liu, Yufan Hu, Lichao Sun, Xiang Li, and Yixuan Yuan. Neuroclaw technical report. *arXiv preprint arXiv:2604.24696*, 2026.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.

Jun Xie, Jing Wang, Xiumei Wu, Xinyuan Liu, Yiqi Mi, Qinjin Liu, Tong Xu, Chen Liu, Huafu Chen, and Jing Guo. Neura: An agentic system for autonomous neuroimaging workflows. *bioRxiv*, pp. 2026–04, 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yongyue Zhang, Michael Brady, and Stephen Smith. Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm. *IEEE Transactions on Medical Imaging*, 20(1):45–57, 2001.

Lujia Zhong, Yihao Xia, Jianwei Zhang, Jiaxin Yue, Mingyang Xia, Yonggang Shi, et al. Neuroagent: Llm agents for multimodal neuroimaging analysis and research. *arXiv preprint arXiv:2605.06584*, 2026.

# A End-to-End Autonomous Neuroimaging Analysis Experiments Details

## A.1 Task Descriptions of the End-to-End Neuroimaging Analysis

We evaluate NEXUS in an end-to-end neuroimaging analysis setting in which the agent is given raw neuroimaging data together with a natural-language task specification. The task defines the prediction target and required deliverables, but leaves the analysis strategy unspecified. The agent must therefore inspect the dataset, develop and execute an appropriate preprocessing and quality-control workflow, perform downstream analysis and model training, and ultimately deliver a reusable analysis package consisting of the finalized preprocessing pipeline, trained model artifacts, and the corresponding inference procedure for held-out evaluation. The two benchmarks instantiate this setting with different modality-specific objectives: ADHD-200 requires functional connectome-based diagnosis prediction from raw fMRI, whereas ADNI requires structural diagnosis prediction from raw T1-weighted MRI using voxel-wise gray matter features.

**ADNI End-to-End Task Prompt**

```
In {ADNI_RAW_DATA_DIR}, there are raw sMRI data from several ADNI
    subjects. Your task is to preprocess these subjects' data, then
    train classification model(s) using voxel-wise gray matter
    probability map features derived from voxel-based morphometry
    (VBM).

The target label is 'diagnosis' (CN vs AD), which can be found in:
{ADNI_RAW_DATA_DIR/participants.tsv}.

Your final deliverables are:

1. A complete neuroimaging preprocessing pipeline.
2. Trained downstream prediction model.
3. The corresponding inference script that can load the trained model
    and produce predictions on the held-out test set.

The provided dataset should be treated as training set. Your delivered
    preprocessing pipeline and model will be applied to another
    held-out test set of subjects (which is invisible to you). Your
    performance will be evaluated based on the prediction score on
    this test set. This is a competitive evaluation. The primary
    objective is to achieve the highest possible score on the held-out
    test set, while ensuring that the workflow remains
    methodologically sound, logically justified, reproducible, and
    free from data leakage or invalid assumptions.

You can explore different preprocessing pipelines (up to three) using
    available neuroimaging tools and design different modeling methods
    to find the one that potentially yields the best prediction
    performance on the held-out test set.

Besides the final deliverables, you should also clearly explain the
    rationale behind the final selection of the preprocessing pipeline
    and the corresponding modeling methods.
```

**ADHD-200 End-to-End Task Prompt**

```
In {ADHD_RAW_DATA_DIR}, there are raw data from several ADHD-200
    subjects. Your task is to preprocess these subjects' data, then
    train classification model(s) for functional brain connectome
    analysis based on the preprocessed data.

The target label is 'diagnosis' (Control vs ADHD), which can be found
    in:
{ADHD-200_RAW_DATA_DIR/participants.tsv}.
```

```
Your final deliverables are:

1. A complete neuroimaging preprocessing pipeline.
2. Trained downstream prediction model.
3. The corresponding inference script that can load the trained
    model(s) and produce predictions on the held-out test set.

The provided dataset should be treated as training set. Your delivered
    preprocessing pipeline and model will be applied to another
    held-out test set of subjects (which is invisible to you). Your
    performance will be evaluated based on the prediction score on
    this test set. This is a competitive evaluation. The primary
    objective is to achieve the highest possible score on the held-out
    test set, while ensuring that the workflow remains
    methodologically sound, logically justified, reproducible, and
    free from data leakage or invalid assumptions.

You can explore different preprocessing pipelines (up to three) using
    available neuroimaging tools and design different modeling methods
    to find the one that potentially yields the best prediction
    performance on the held-out test set.

Besides the final deliverables, you should also clearly explain the
    rationale behind the final selection of the preprocessing pipeline
    and the corresponding modeling methods.
```

### A.2 Baseline Workflow Specifications

**Preprocessing pipelines:**

- **fMRIPrep for ADHD:** The fMRIPrep-based baseline for ADHD uses the standard participant-level fMRIPrep workflow with default preprocessing logic and no task-specific customization beyond specifying the target output space and runtime resources. For each subject, the raw BIDS-formatted dataset is processed with the standard fMRIPrep pipeline for both structural MRI and functional MRI. Preprocessing outputs are generated in the `MNI152NLin2009cAsym` space. In this baseline, we use the preprocessed BOLD images in MNI space as the downstream input for functional connectome analysis. No additional custom preprocessing steps are inserted outside the standard fMRIPrep workflow.
- **AFNI for ADHD:** fMRI data are preprocessed with AFNI 26.0.08 using a fixed six-stage workflow. For each BOLD run, slice-timing correction is first performed with `3dTshift`; the slice acquisition pattern is inferred from the BIDS metadata when possible and otherwise defaults to `alt+z`. Motion correction is then applied with `3dvolreg`, using the first volume as the registration base, and motion parameters are retained for later nuisance regression. For each subject, the T1-weighted image is normalized to the MNI152 2009 template space using `@SSwarper`. The motion-corrected fMRI run is subsequently normalized to MNI space with AFNI's `afni_proc.py`-based volumetric alignment and nonlinear warping procedure, using the normalized T1w image as anatomical support. After normalization, the fMRI time series is spatially smoothed with a 2.0 mm FWHM Gaussian kernel using `3dmerge`, followed by temporal filtering and nuisance regression with `3dTproject` using a 0.01–0.1 Hz band-pass range and polynomial detrending of order 2. The resulting temporally filtered fMRI time series in MNI space serves as the input for downstream functional connectome analysis.
- **FSL for ADHD:** Resting-state fMRI is preprocessed with FSL 6.0.7 (Jenkinson et al., 2012) following the standard volume-based functional pipeline. Each subject's 4D BOLD series is rigid-body motion-corrected with MCFLIRT (Jenkinson et al., 2002), the mean reference volume is linearly registered to the MNI152 2 mm template via FLIRT (Jenkinson et al., 2002) using a 12-degree-of-freedom affine transform, and the linear estimate is refined to a nonlinear warp with FNIRT under the `T1_2_MNI152_2mm` configuration. The composite warp is propagated to every volume of the realigned 4D series via `applywarp`,

yielding a per-subject BOLD time series resampled into MNI152 2 mm space, which serves as the input to downstream connectivity analysis.

- **SPM for ADHD:** Resting-state fMRI is preprocessed with SPM12 (Penny et al., 2011) using the canonical EPI normalization workflow. After slice-timing correction, all volumes are realigned to the within-run mean image to remove head-motion artifacts. The mean functional image is then spatially normalized to the MNI template via SPM's unified segmentation–normalization, and the estimated deformation field is applied to the full realigned 4D series. The warped time series is smoothed with an isotropic Gaussian kernel (FWHM 5 mm), producing the standard `swra*` outputs in MNI space that are passed to the downstream connectivity analysis.
- **fMRIPrep for ADNI:** The fMRIPrep-based baseline for ADNI uses the standard participant-level fMRIPrep workflow in anatomical-only mode. For each subject, the raw BIDS-formatted T1-weighted MRI is processed using the default fMRIPrep anatomical preprocessing logic without additional custom modifications. The pipeline produces bias-corrected native-space T1w images, native-space brain masks, native-space tissue segmentation maps, and normalized T1w images in `MNI152NLin2009cAsym` space. For downstream structural analysis, we use the gray matter probability map in MNI space produced by fMRIPrep as the voxel-wise VBM-style feature representation. Native-space masks and segmentation outputs are retained only as supporting derivatives and are not additionally modified outside the standard workflow.
- **AFNI for ADNI:** T1-weighted images are preprocessed with AFNI 26.0.08 using a fixed structural MRI workflow. Each subject's T1w image is first brain-extracted with `3dSkullStrip`, and the skull-stripped image is then segmented into CSF, gray matter, and white matter with `3dSeg`. The original T1w image is normalized to the MNI152 2009 template space using `@SSwarper`, which produces the native-to-template transforms. These transforms are then applied with `3dNwarpApply` to the native-space gray matter probability map, yielding a voxel-wise GM probability volume in MNI space that serves as the input feature representation for downstream structural classification.
- **FSL for ADNI:** T1-weighted images are preprocessed with FSL 6.0.7 (Jenkinson et al., 2012). Each subject's T1w volume is brain-extracted using BET (Smith, 2002) with fractional intensity threshold 0.4 and segmented into gray matter, white matter, and CSF partial-volume estimates with FAST (Zhang et al., 2001). The brain-extracted image is linearly registered to the MNI152 2 mm template using FLIRT (12-DOF affine), and a nonlinear warp from native T1w space to MNI is estimated with FNIRT under the `T1_2_MNI152_2mm` configuration. The estimated warp is finally applied via `applywarp` to the gray matter probability map (`pve_1`), yielding a voxel-wise GM probability volume in MNI space that serves as the input feature representation for downstream classification.
- **ANTs for ADNI:** T1-weighted images are preprocessed with ANTs 2.6.5 (Avants et al., 2008; 2011a). Each subject's T1w volume undergoes N4 bias-field correction (Tustison et al., 2010), followed by template-based brain extraction with `antsBrainExtraction.sh` (Avants et al., 2011b) using the MNI152 brain template and probability mask. Three-tissue segmentation (CSF, GM, WM) is then performed by the iterative N4–Atropos procedure (`antsAtroposN4.sh`) (Avants et al., 2011b), which alternates intensity inhomogeneity correction and posterior estimation to produce stable tissue probability maps. The brain-extracted image is nonlinearly registered to the MNI152 1 mm template with `antsRegistrationSyNQuick.sh` (rigid + affine + SyN diffeomorphism) (Avants et al., 2008), and the GM posterior map is warped to MNI space via `antsApplyTransforms` to yield the voxel-wise GM probability volume used by downstream classification.

**Automated downstream analysis for baselines:**

- **Downstream analysis baseline for ADHD:** For all ADHD baselines, downstream analysis starts from the preprocessed fMRI time series produced by the corresponding preprocessing backbone. Subject-level functional connectivity matrices are first constructed under three candidate parcellation atlases, namely AAL90 (Tzourio-Mazoyer et al., 2002), Schaefer200 (Schaefer et al., 2017), and HCP360 (Glasser et al., 2016), using correlation-

based functional connectivity. Model selection is then performed by stratified 5-fold cross-validation on the training set over the joint space of atlas choice, model class, and hyperparameters. Candidate models include BrainNetworkTransformer (Kan et al., 2022) (a state-of-the-art graph transformer model for functional brain connectome analysis), NeuroGraph (Said et al., 2023) (a state-of-the-art graph neural network model for functional brain connectome analysis with residual connections), logistic regression, elastic-net logistic regression, and SVM. For the classical models, the upper-triangular connectivity entries are vectorized as features, followed by univariate two-sample $t$-test feature selection, standardization, and classifier fitting. The number of retained features is chosen adaptively for each atlas and fold as $\{1, 2, 3, 5, 10\} \times N_{\text{fold}}$, where $N_{\text{fold}}$ denotes the number of training subjects in the corresponding cross-validation fold. Logistic regression searches over both $L_2$- and $L_1$-regularized models with $C$ values spanning $10^{-4}$ to $3 \times 10^2$. Elastic-net searches over $C$ values from $3 \times 10^{-4}$ to 10, together with $\texttt{l1_ratio}$ values ranging from 0.1 to 0.9. SVM searches over both linear and RBF kernels, with linear-kernel $C$ values spanning $10^{-4}$ to $10^2$ and RBF settings covering both `scale`/`auto` and explicit numeric $\gamma$ values. For graph-based models, the original connectivity matrix is used as the node-feature matrix, while NeuroGraph additionally uses a sparsified connectivity matrix as adjacency. BrainNetworkTransformer searches over 1–2 transformer layers, 4–24 ROI clusters in the larger search space, hidden size 32–256, attention heads in {2, 4, 5, 10}, learning rates from $2 \times 10^{-4}$ to $1.5 \times 10^{-3}$, weight decay from $2 \times 10^{-3}$ to $10^{-1}$, batch size 4–16, and 12–35 training epochs. NeuroGraph searches over hidden channels 8–48, 1–2 graph layers, hidden dimension 16–192, learning rates from $2 \times 10^{-4}$ to $1.5 \times 10^{-3}$, weight decay from $2 \times 10^{-3}$ to $10^{-1}$, batch size 4–16, and 15–30 epochs. All baseline results reported in the paper use mean cross-validated ACC on the validation set as the final selection criterion. After selection, the chosen atlas-model-hyperparameter configuration is retrained on the full training set and evaluated exactly once on the held-out test set.

- **Downstream analysis baseline for ADNI:** For all ADNI baselines, downstream analysis starts from the gray matter probability maps produced by the corresponding structural preprocessing backbone. Each subject image is aligned to a common gray matter mask in MNI space, after which two feature representations are extracted: (i) masked voxel-wise gray matter features for classical machine learning models, and (ii) masked 3D gray matter volumes for volumetric CNN modeling. Model selection is performed by stratified 5-fold cross-validation on the training set over model classes and hyperparameters. Candidate models include logistic regression, elastic-net logistic regression, SVM, XGBoost, and 3D CNN. For the classical models and XGBoost, feature-retention ratio is searched jointly with model hyperparameters. Across all attempted configurations, logistic regression searches over both $L_2$- and $L_1$-regularized models with `feature_keep_ratio` values in the range 0.003–0.25 and $C$ values in the range 0.02–1.0. Elastic-net searches over `feature_keep_ratio` values from 0.003 to 0.25, $C$ values from 0.005 to 0.5, and $\texttt{l1_ratio}$ values ranging from 0.3 to 0.95. SVM searches over both linear and RBF kernels, with `feature_keep_ratio` values from 0.003 to 0.25, linear-kernel $C$ values from 0.03 to 1.0, and RBF configurations including both `scale` and `auto` $\gamma$ settings. XGBoost searches over `feature_keep_ratio` values from 0.01 to 0.1; the tree-based hyperparameters span 30–380 trees, maximum depth 1–3, learning rate 0.004–0.035, subsample 0.45–0.95, column-sampling ratio 0.2–0.9, and additional regularization parameters including $\lambda$, $\alpha$, $\gamma$, and `min_child_weight`. The 3D CNN baseline operates directly on masked 3D gray matter volumes and searches over base channels 2–14, dropout 0.4–0.75, learning rates from $1.5 \times 10^{-4}$ to $10^{-3}$, weight decay from $4 \times 10^{-4}$ to $3 \times 10^{-3}$, batch size 2–12, and 13–26 epochs. All baseline results reported in the paper use mean cross-validated ACC on the validation set as the final model-selection criterion. After selection, the chosen model is retrained in the final training stage on the full training set and is then evaluated exactly once on the held-out test set.

**Exhaustive workflow search baseline.** In addition to the standard workflow baselines above, we construct an *Exhaustive Workflow Search* baseline that extends automated selection from downstream modeling to the joint workflow space of preprocessing, quality control, and downstream modeling. All candidate workflows are selected using the training set only, and the held-out test set remains inaccessible throughout the search process.

The search space consists of the following three dimensions:

- **Preprocessing pipeline.** For ADHD-200, the candidate preprocessing pipelines are {fMRIPrep, AFNI, FSL, SPM}. For ADNI, the candidates are {fMRIPrep, AFNI, FSL, ANTs, ANTs+FSL hybrid}. The ANTs+FSL hybrid is a cross-backend pipeline composed by NEXUS during its autonomous workflow exploration and is additionally included in the exhaustive-search space so that the exhaustive baseline's search space strictly contains the workflow configurations explored by NEXUS and is evaluated with a larger search budget.
- **Quality-control strategy.** We search over {no QC, metric-then-visual QC}. The latter uses the same quantitative screening, visual-inspection primitives, and QC criteria available to NEXUS, thereby allowing the exhaustive baseline to explicitly search over whether the QC stage is included in the final workflow.
- **Downstream modeling configuration.** The candidate modeling space is the union of the automated model-selection spaces described above and all valid modeling configurations appearing in the NEXUS execution trajectories that produced the delivered models across the five runs of each dataset. This construction ensures that the exhaustive baseline has access to every downstream configuration explored by NEXUS in these trajectories, in addition to the predefined AutoML search space.

Taking the Cartesian product of these dimensions yields 33,008 candidate workflow configurations for each ADHD-200 repetition ($4$ preprocessing pipelines $\times$ $2$ QC choices $\times$ $4{,}126$ modeling configurations) and 11,470 configurations for each ADNI repetition ($5$ preprocessing pipelines $\times$ $2$ QC choices $\times$ $1{,}147$ modeling configurations). This search space is more generous than the workflow exploration performed by NEXUS. For each exhaustive-search repetition, candidate workflows are evaluated using the same training-only cross-validation protocol as the standard baselines. The configuration with the highest mean cross-validated ACC is selected, refit using the full training set, and evaluated exactly once on the held-out test set.

### A.3 Component-wise Analysis of Agentic Decision-Making

The end-to-end performance of NEXUS results from decisions made across multiple stages of the analysis workflow. To better understand where agentic decision-making contributes, we separately examine three major decision spaces: preprocessing pipeline construction, quality control, and downstream modeling optimization. We evaluate their contributions through component-specific ablations while keeping the remaining workflow as unchanged as possible.

**Quality control and downstream optimization.** We first ablate the QC and downstream-analysis decisions starting from the workflow delivered by NEXUS. In **w/o QC**, all subjects excluded by the QC Agent are reintroduced, while retaining the NEXUS-delivered preprocessing pipeline and downstream modeling procedure. In **w/o Agentic Modeling Optimization**, we retain the NEXUS-delivered preprocessing pipeline and QC-filtered cohort, but replace the Downstream Analysis Agent's adaptive modeling decisions with the same automated model and hyperparameter search used by the standard baselines. Table 3 summarizes the results. QC contributes clearly on ADHD-200, where removing it lowers all three metrics. This is expected for fMRI, which is noisier and of more variable quality than structural T1w, so the QC Agent excludes more subjects (about 30 of 160 per run on average). On ADNI the images are of consistently high quality and QC excludes far fewer subjects (10 to 20 per run), so its effect is correspondingly smaller and appears as a modest drop in F1 and ACC. The magnitude of the QC effect therefore tracks how much quality variation a dataset actually contains, and QC remains beneficial in both cases. Replacing agentic downstream modeling degrades performance on both datasets.

**Preprocessing pipeline construction.** We next examine decisions in the preprocessing space. NEXUS constructs its preprocessing workflow by composing domain-specific primitives rather than only selecting a single predefined pipeline. In many runs, the resulting workflow remains close to an established single-backend pipeline already represented among our baselines. On ADNI, however, NEXUS additionally orchestrates a cross-backend ANTs+FSL hybrid pipeline that does not correspond to the standard preprocessing workflows. This provides a controlled case for evaluating whether agentic preprocessing composition can produce more informative derivatives.

Table 3: Component-wise ablation of quality-control and downstream-optimization decisions. Results report held-out test performance (%) over five runs.

| | ADHD | | | ADNI | | |
|---|---|---|---|---|---|---|
| **Variant** | **AUC** | **F1** | **ACC** | **AUC** | **F1** | **ACC** |
| Full NEXUS | $66.45 \pm 4.03$ | $65.70 \pm 3.01$ | $64.50 \pm 3.71$ | $93.00 \pm 1.94$ | $82.83 \pm 2.46$ | $84.50 \pm 2.09$ |
| w/o QC | $60.20 \pm 8.56$ | $57.23 \pm 6.33$ | $55.50 \pm 4.81$ | $93.35 \pm 1.71$ | $81.96 \pm 2.91$ | $83.50 \pm 2.85$ |
| w/o Agentic Modeling Optimization | $57.80 \pm 9.29$ | $64.14 \pm 5.64$ | $51.50 \pm 3.35$ | $92.80 \pm 3.70$ | $80.16 \pm 2.45$ | $82.50 \pm 1.77$ |

Table 4: ADNI performance (%) under different preprocessing pipelines with the same automated downstream analysis.

| **Preprocessing + DS** | **AUC** | **F1** | **ACC** |
|---|---|---|---|
| fMRIPrep + DS | $90.60 \pm 1.62$ | $81.06 \pm 2.40$ | $83.00 \pm 1.87$ |
| AFNI + DS | $89.30 \pm 0.46$ | $79.56 \pm 0.89$ | $82.00 \pm 1.00$ |
| FSL + DS | $80.90 \pm 0.90$ | $69.50 \pm 3.90$ | $71.00 \pm 3.70$ |
| ANTs + DS | $89.30 \pm 0.50$ | $75.10 \pm 3.90$ | $76.50 \pm 3.40$ |
| ANTs+FSL Hybrid + DS | $93.30 \pm 1.01$ | $80.82 \pm 2.89$ | $83.00 \pm 2.09$ |

To isolate the preprocessing choice, we compare this hybrid pipeline against the standard preprocessing backbones under the same automated downstream search and without QC filtering. The results are shown in Table 4. The agent-composed hybrid pipeline yields the best AUC of all backbones. This shows that NEXUS does not merely transplant a standard pipeline, but is able to select and recombine processing steps across software backends according to the data and the downstream objective, and that such agentic orchestration can yield derivatives more informative for the downstream task than established single-software stack.

Taken together, these analyses indicate that the end-to-end behavior of NEXUS is not attributable to a single decision stage. QC has its clearest effect on the noisier fMRI task, adaptive downstream optimization contributes on both datasets, and preprocessing composition provides an additional benefit in the ADNI case where NEXUS constructs a cross-backend workflow.

### A.4 Computational Cost Analysis

We provide a detailed cost breakdown for the five independent NEXUS runs reported in the main RQ1 experiment. Table 5 reports the LLM token usage, API cost, wall-clock runtime, and CPU consumption of each run. CPU consumption is measured in core-hours and therefore reflects both the duration and degree of parallelism of the underlying neuroimaging computation.

To further characterize where the computational resources are spent, Table 6 decomposes the average CPU consumption by workflow stage. Most computational consumption comes from subject-level neuroimaging preprocessing, accounting for 97.2% of the CPU cost on ADHD-200 and 89.9% on ADNI. In contrast, agent orchestration itself accounts for only 0.8% and 5.1% of the CPU consumption, respectively. Thus, the dominant computational cost arises from executing the underlying neuroimaging operations rather than from the agentic orchestration layer. Since subject-level processing can be parallelized, additional CPU resources can substantially reduce the wall-clock time without changing the analysis procedure.

### A.5 Example Running Trajectory of End-to-End Neuroimaging Analysis by NEXUS

To further illustrate how NEXUS operates beyond its final predictive performance, we present an example end-to-end execution trajectorie out of 5 independent runs on ADHD-200 and ADNI respectively. These trajectories make explicit that the system does not follow a fixed pre-scripted pipeline. Instead, it begins from a high-level task objective, inspects dataset-specific conditions, performs early-stage quality control, constructs candidate preprocessing and analysis workflows, and then revises its strategy based on intermediate observations and runtime feedback. Across both

Table 5: Detailed per-run computational cost of NEXUS in the RQ1 experiments. Prompt and completion tokens correspond to `gpt-5.2` usage. API cost reports the corresponding LLM API cost.

| Dataset | Run | Prompt Tokens | Completion Tokens | API Cost | Wall-clock / CPU Cost |
|---|---|---|---|---|---|
| ADHD-200 | 1 | 2.55M | 165.1K | $6.77 | 58.00 h / 4,201 core-h |
| | 2 | 3.07M | 137.6K | $7.30 | 54.65 h / 4,014 core-h |
| | 3 | 5.44M | 172.8K | $11.94 | 60.55 h / 4,149 core-h |
| | 4 | 2.96M | 144.6K | $7.21 | 27.79 h / 1,251 core-h |
| | 5 | 5.21M | 192.1K | $11.81 | 70.93 h / 4,862 core-h |
| | Mean | 3.85M | 162.4K | $9.01 | 54.4 h / 3,696 core-h |
| ADNI | 1 | 4.70M | 175.4K | $10.68 | 83.21 h / 2,712 core-h |
| | 2 | 2.62M | 140.5K | $6.56 | 26.92 h / 1,499 core-h |
| | 3 | 1.24M | 102.0K | $3.61 | 27.60 h / 1,459 core-h |
| | 4 | 2.05M | 118.9K | $5.26 | 28.69 h / 1,499 core-h |
| | 5 | 2.51M | 128.7K | $6.19 | 23.67 h / 1,304 core-h |
| | Mean | 2.63M | 133.1K | $6.46 | 38.0 h / 1,695 core-h |

Table 6: Average CPU consumption per run by workflow stage. Percentages denote the fraction of total CPU core-hours.

| Stage | ADHD-200 | ADNI |
|---|---|---|
| Preprocessing | 3,592 core-h (97.2%) | 1,524 core-h (89.9%) |
| Quality control | 67 core-h (1.8%) | 48 core-h (2.8%) |
| Downstream modeling | 9 core-h (0.2%) | 37 core-h (2.2%) |
| Agent orchestration | 29 core-h (0.8%) | 86 core-h (5.1%) |

datasets, the trajectories show several characteristic forms of agentic behavior: dynamic decomposition of the end-to-end problem into coordinated sub-tasks, self-recovery from unexpected issues, iterative comparison and refinement of alternative preprocessing pipelines, hierarchical QC-gated decision points, and downstream modeling refinement. Table 7 shows the example ADHD-200 run, and Table 8 shows an example ADNI run.

Table 7: Example running trajectory of NEXUS on ADHD-200 from one of five runs.

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 1 | First understand the ADHD-200 data organization, labels, and site heterogeneity, then decide subsequent QC / preprocessing / modeling strategies. | `Data Awareness Agent` | Inspected the directory structure, the BIDS-likeness, and the per-subject T1w and resting-state BOLD file patterns; read `participants.tsv` and representative sidecar JSONs. | Confirmed the dataset is single-session, BIDS-like multi-site ADHD-200 data; 160 subjects in total, with `diagnosis` as the label column and `site` information available; TR, volume counts, and scan parameters differ across sites. |

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 2 | Raw-data QC must be performed before any preprocessing, to prevent low-quality data from entering downstream stages. | `Quality Control Agent` | Ran MRIQC on all raw T1w and BOLD scans, performed group-level outlier screening, and conducted visual QC only on flagged individuals. | Effectively evaluated 158 subjects, yielding `PASS=134` and `FAIL=24`; the 24 raw-QC fail subjects were excluded, with common failure reasons being BOLD dropout/ghosting/motion and T1w ringing/noise/inhomogeneity. |
| 3 | First trial-run an initial `pipeline 1_A` on a small set of PASS subjects to verify whether preprocessing is usable. | `Processing Agent` | Performed a sample run of `pipeline 1_A = fMRIPrep` on 10 PASS subjects and checked whether the key MNI outputs were present. | All 10 sample subjects completed; MNI preprocessed BOLD, mean BOLD, and brain mask were all present. |
| 4 | Check the MNI normalization quality of `pipeline 1_A` at the sample level first, before deciding whether to scale to the full cohort. | `Quality Control Agent` | Ran visual QC on the fMRIPrep MNI mean BOLD + brain mask of the 10 sample subjects, checking only normalized-to-MNI quality. | All 10/10 passed, indicating that the MNI normalization quality of `pipeline 1_A` is acceptable on the sample. |
| 5 | After sample QC passed, scale `pipeline 1_A` to all subjects not excluded by raw-QC. | `Processing Agent` | Scaled `pipeline 1_A` to all subjects not excluded by raw-QC, and checked the availability of MNI BOLD, mean and mask files. | Processed 136 subjects in total with 0 processing errors; MNI BOLD / mean / mask were complete for all 136 subjects. |
| 6 | After full-cohort preprocessing, processed-data QC should be repeated to form the final modeling cohort. | `Quality Control Agent` | Computed MNI QC metrics (NMI, Dice) on all 136 `pipeline 1_A` outputs, flagged the bottom 15%, and ran visual QC on the flagged subjects. | All 136 subjects had complete files; 40 were flagged for visual review; in the end only `sub-1057962` was judged FAIL due to MNI size mismatch, so the modeling-ready cohort from `pipeline 1_A` is 135 subjects. |
| 7 | Conduct connectome-based diagnosis classification on the QC-validated `pipeline 1_A` cohort, prioritizing site generalization. | `Downstream Analysis Agent` | Built connectome features for the 135 QC-passed subjects and adaptively refined downstream modeling: fixed a PCA fold-size failure by revising the CV script, expanded validation across site-grouped/site-covariate CV settings, compared classical ML with a BNT trial, and rejected BNT after observing fold-level instability. | Yielded a final modeling cohort of `n=135`; the best configuration is HCP360 + Fisher-z + upper-triangle + StandardScaler + PCA(100) + balanced Logistic Regression; under site-grouped CV the performance is approximately `AUROC=0.607`, `F1=0.518`, `ACC=0.585`; the two LR models (`C=0.1` and `C=1.0`) were saved together with `predict.py`. |
| 8 | Building on the existing `pipeline 1_A` modeling result, explore a second preprocessing route to test whether generalization can be improved. | `Processing Agent` | Built `pipeline 2_A` and ran FSL-based preprocessing on 10 sample subjects: motion correction, func-to-anat, func-to-MNI, 2 mm smoothing, and 0.01–0.1 Hz temporal filtering. | All 10 subjects were processed successfully; subject-space coreg outputs, MNI outputs, and final filtered MNI 4D timeseries were generated and ready for QC. |

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 9 | Conduct two-stage QC on the FSL sample pipeline to verify whether it is worth scaling to the full cohort. | `Quality Control Agent` | Ran Stage A (func-to-anat) and Stage B (func-to-MNI) visual QC on the 10 subjects from `pipeline 2_A`. | Stage A `PASS=10/10`; Stage B `FAIL=10/10`, with all samples showing a consistent MNI-template-vs-EPI size/alignment mismatch, suggesting a systematic normalization issue, so `pipeline 2_A` was rejected. |
| 10 | After the systematic failure of the FSL route, try the more robust AFNI-based alternative `pipeline 2_B`. | `Processing Agent` | On the same 10 subjects, built `pipeline 2_B`: AFNI T1w-to-MNI, AFNI fMRI-to-MNI, 2 mm smoothing, bandpass filtering, and produced the files required for alignment and normalization QC. | All 10 subjects were processed successfully; T1w-space alignment files, MNI normalization files, motion regressors, and the final connectivity-ready filtered MNI 4D timeseries were generated. |
| 11 | Conduct two-stage QC on the AFNI sample pipeline to judge whether it outperforms FSL and whether it is worth replacing fMRIPrep. | `Quality Control Agent` | Ran Stage A (fMRI-to-anatomical) and Stage B (fMRI-to-MNI) visual QC on the 10 subjects from `pipeline 2_B`. | Stage A `PASS=9/10`, with only `sub-0010048` failing; Stage B `PASS=9/10`, with only `sub-0010029` failing; AFNI clearly outperforms FSL but is still less robust than fMRIPrep at scale. |
| 12 | Integrate raw QC, processed QC, downstream modeling performance, and the alternative pipelines' sample-QC results to choose the final submission. | `Supervisor Agent` | Aggregated end-to-end evidence, compared `pipeline 1_A`, `pipeline 2_A`, and `pipeline 2_B` in terms of reliability, scalability, and modeling usability, and finalized the deliverables. | Selected **`Pipeline 1_final = fMRIPrep + stringent QC exclusion`**; raw QC excluded 24 subjects, processed QC further excluded `sub-1057962`, leaving a final modeling cohort of 135 subjects; the connectome choice is HCP360 + Fisher-z, the final model is a PCA100 + balanced Logistic Regression two-model average, and the inference script is `models/predict.py`. |

Table 8: Example running trajectory of NEXUS on ADNI from one of five runs.

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 1 | Before doing anything else, understand the ADNI dataset structure, label distribution, and available image paths so the rest of the workflow can be planned. | Data Awareness Agent | Scanned the BIDS-like directory tree under `ADNI_RawData`, inspected `participants.tsv`, and confirmed the T1w naming convention as well as the diagnosis label encoding. | Confirmed `160` subjects in total, `CN=80` and `AD=80`, all with `ses-1` T1w scans, providing a solid basis for structural QC and downstream modeling. |

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 2 | Run quality control on raw T1w data before any preprocessing so that obviously low-quality subjects are excluded up front. | Quality Control Agent | Ran MRIQC on all T1w scans, used group-level outlier screening to flag abnormal subjects, performed targeted visual QC only on those flagged subjects, and aggregated per-subject PASS/FAIL decisions. | All `160/160` subjects went through MRIQC; final verdict was `157 PASS / 3 FAIL`, excluding `sub-033S1087`, `sub-126S0784`, and `sub-941S1194`. All three were excluded due to severe raw-T1w artifacts (motion ringing/ghosting, susceptibility-related signal dropout, and high noise). |
| 3 | Test a strong baseline structural pipeline on a small sample first to produce VBM-style GM probability maps in MNI space. | Processing Agent | Ran `Pipeline 1_A` on 10 sampled subjects: ANTs N4 → ANTs skull stripping → FSL FAST → ANTs MNI normalization → ANTs warp of the GM probability map to MNI. | All 10 sampled subjects completed successfully with all key derivatives present, yielding MNI-space GM probability maps ready for QC. |
| 4 | Before scaling to the full cohort, rigorously check the sample-level processed data quality of `Pipeline 1_A`. | Quality Control Agent | Completed skull-stripping visual QC on the 10 sampled subjects; then, because the exact input files for tissue segmentation QC and normalization QC needed supervisor confirmation, paused and asked the supervisor to confirm which derivatives to use. | Identified `sub-003S4892` as a skull-stripping failure; QC was paused awaiting confirmation that `fast_seg`/`fast_pveseg` and `norm_ants_Warped` should be used as inputs for the remaining QC steps. |
| 5 | After clarifying the QC inputs, complete all sample-level processed-data QC for `Pipeline 1_A`. | Quality Control Agent | Per supervisor confirmation, finished tissue segmentation QC and T1-to-MNI normalization QC, and aggregated the three-step decisions into final per-subject verdicts. | `Pipeline 1_A` sample QC ended at `2/10` failures: `sub-003S1021` failed tissue segmentation and `sub-003S4892` failed skull stripping; the other 8 subjects passed. |
| 6 | Given the issues exposed by sample QC, attempt one alternative preprocessing variant. | Processing Agent | Ran `Pipeline 1_B` on the same 10 sampled subjects: stronger N4 + AFNI skull stripping + FSL FAST + ANTs normalization/warp. | `Pipeline 1_B` finished on all 10 subjects with no crashes and complete derivatives, ready for comparative QC. |
| 7 | Compare sample-level QC performance between `Pipeline 1_A` and `Pipeline 1_B` to choose the final structural pipeline. | Quality Control Agent | Performed skull stripping, tissue segmentation, and normalization QC on the 10 `Pipeline 1_B` subjects and compared the results against the `Pipeline 1_A` outcomes. | `Pipeline 1_B` was clearly worse — `8/10` failures, mostly due to over-aggressive AFNI skull stripping — so `Pipeline 1_A` was judged superior. |
| 8 | Since `Pipeline 1_A` won the sample QC comparison, lock it in as the final preprocessing pipeline and scale to all eligible subjects. | Processing Agent | Promoted `Pipeline 1_A` to `Pipeline 1_final` and ran it at scale on all `157` raw-QC-passed subjects. | All `157/157` subjects processed successfully with no missing key derivatives or crashes, producing whole-cohort MNI-space GM probability maps. |

| # | Supervisor Thought | Active Sub-Agent | Action Performed | Outcome |
|---|---|---|---|---|
| 9 | After full-cohort preprocessing, run processed-data QC again to ensure only high-quality samples enter modeling. | Quality Control Agent | On the 157 `pipeline_1_final` subjects, performed step-specific metric screening (flagging the most abnormal $\sim$15% per step) followed by step-specific visual QC, and aggregated final PASS/FAIL decisions. | Final outcome: `144 PASS / 13 FAIL`; the 13 processed-QC failures were excluded, leaving the final modeling cohort. Most exclusions were caused by fragmented/noisy GM-WM tissue segmentation, with `sub-041S1435` being the only subject failing all three QC steps. |
| 10 | Using the final QC-passed cohort, train a CN-vs-AD classifier and produce a reusable inference script. | Downstream Analysis Agent | Extracted voxel-wise GM features, ran leakage-free 5-fold CV, iteratively explored multiple model families and refinements; here refinement specifically refers to adjusting the feature-selection percentile, regularization strength, PCA variants, and elastic-net hyperparameters based on the initial CV results, then comparing AUROC/ACC/F1 across rounds before retraining the best configuration on the full training set and saving `predict.py`. | Selected the final model on `144` subjects: `elastic-net logistic regression (SAGA)` with 5-fold CV `AUROC = 0.913` $\pm$ `0.076`; the trained model and inference script were saved. |
| 11 | Aggregate and deliver the final neuroscientist agent outputs. | Supervisor Agent | Consolidated raw QC, the final preprocessing pipeline, processed-data QC, downstream modeling, and the inference deliverables into one summary. | Final deliverables: full structural derivatives from `Pipeline 1_final`, the `144`-subject modeling cohort, the trained CN-vs-AD classifier, and the `predict.py` inference script. |

# B Architectural Ablations Complementary Contents

## B.1 Error Taxonomy in the Ablation Study

To better interpret the failure patterns observed in the ablation study, we categorize execution errors into the following types:

- **Tool / Primitive Misuse.** Errors arising when an agent incorrectly invokes a provided domain-specific primitive or general-purpose tool (e.g., `run_bash_command`), including wrong arguments, schema mismatch, invalid input types, or protocol-level misuse of the calling interface.
- **Code Generation / Execution Error.** Failures caused by agent-written Python or Bash code, including syntax errors, incompatible APIs, invalid package usage, shape or numerical mismatches, incorrect parameters, and other runtime bugs. This category also includes implicit logic errors, where the code executes without crashing but implements an incorrect procedure.
- **State Drift.** Errors caused by incorrect propagation of execution state across stages, leading to inconsistencies between earlier and later decisions. Examples include reintroducing subjects that were already excluded during raw-data QC, repeating completed steps unnecessarily.

Table 9: Detailed ablation records on ADHD-200. Runtime is reported in hours. Prompt and completion tokens are abbreviated in M/K units. TO denotes timeout. N/A indicates this run failed completely, so the timing statistics are not meaningful.

| Variant | Run | Completion | Runtime (h) | Prompt Tok | Comp Tok | $ API Cost | # Errors | # Recovery |
|---|---|---|---|---|---|---|---|---|
| Full system | 1 | ✓ | 5.54 | 1.94M | 100.89K | 4.81 | 1 | 1 |
| Full system | 2 | ✓ | 7.56 | 7.81M | 176.29K | 16.14 | 0 | 0 |
| Full system | 3 | ✓ | 8.35 | 1.66M | 101.92K | 4.34 | 2 | 2 |
| Full system | 4 | ✓ | 3.74 | 2.65M | 107.09K | 6.13 | 3 | 3 |
| Full system | 5 | ✓ | 11.62 | 1.87M | 110.12K | 4.81 | 0 | 0 |
| Single-Agent | 1 | ✓ | 8.81 | 2.20M | 22.66K | 4.16 | 3 | 1 |
| Single-Agent | 2 | ✓ | 9.58 | 2.34M | 24.03K | 4.43 | 2 | 2 |
| Single-Agent | 3 | ✓ | 8.59 | 3.62M | 25.24K | 6.68 | 4 | 2 |
| Single-Agent | 4 | × | N/A | 719.80K | 5.58K | 1.34 | 10 | 2 |
| Single-Agent | 5 | ✓ | 8.52 | 2.14M | 21.89K | 4.05 | 3 | 3 |
| w/o Primitive Abstraction | 1 | ✓ | 10.58 | 2.04M | 96.29K | 4.91 | 5 | 3 |
| w/o Primitive Abstraction | 2 | × | TO | 470.71K | 27.51K | 1.21 | 7 | 1 |
| w/o Primitive Abstraction | 3 | × | TO | 1.33M | 54.49K | 3.10 | 10 | 1 |
| w/o Primitive Abstraction | 4 | × | TO | 1.17M | 65.85K | 2.98 | 14 | 6 |
| w/o Primitive Abstraction | 5 | × | TO | 528.40K | 48.98K | 1.61 | 11 | 5 |
| w/o Code-Centric Execution | 1 | × | TO | 2.04M | 96.29K | 4.91 | 4 | 1 |
| w/o Code-Centric Execution | 2 | × | TO | 5.72M | 180.55K | 12.54 | 5 | 2 |
| w/o Code-Centric Execution | 3 | × | TO | 1.24M | 102.85K | 3.61 | 8 | 2 |
| w/o Code-Centric Execution | 4 | × | TO | 3.55M | 99.31K | 7.61 | 6 | 1 |
| w/o Code-Centric Execution | 5 | × | TO | 6.89M | 136.35K | 13.97 | 5 | 2 |
| w/o JIT Context Injection | 1 | ✓ | 5.53 | 1.94M | 105.06K | 4.86 | 5 | 5 |
| w/o JIT Context Injection | 2 | ✓ | 5.73 | 1.78M | 106.94K | 4.62 | 1 | 1 |
| w/o JIT Context Injection | 3 | ✓ | 11.42 | 4.65M | 147.36K | 10.19 | 3 | 0 |
| w/o JIT Context Injection | 4 | ✓ | 9.53 | 2.44M | 113.53K | 5.85 | 3 | 2 |
| w/o JIT Context Injection | 5 | ✓ | 9.43 | 2.74M | 97.33K | 6.15 | 9 | 4 |

- **Instruction Violation.** Failures in which the agent does not follow explicitly specified system-level or prompt-level requirements. This includes violating prescribed protocols, skipping required components of a procedure, or ignoring mandated tools and instead using unsupported or self-invented alternatives.
- **Scalability / Resource Failure.** Failures caused by poor scalability or resource management, such as overly serial execution, exhaustion of the wall-clock budget, hitting the ReAct recursion or action limit, out-of-memory errors.
- **Hallucination / Fabrication.** Failures in which the agent invents nonexistent data, fabricates progress or completion status, reports results that were not actually produced, or makes unsupported recovery attempts based purely on intuition rather than evidence from the execution context.
- **File Path Error.** Errors involving incorrect file paths, missing path resolution, or mismatched assumptions about file locations. These are especially common in long neuroimaging workflows with complex dataflow and many intermediate derivatives.
- **Workflow Orchestration Error.** Higher-level failures in workflow organization, such as performing steps in an incorrect order, executing stages before prerequisites are complete, confusing multi-subject execution states.

### B.2 Case Studies of Run-Time Failures and Incorrect Decisions

To complement the aggregate error statistics, we further examine representative run-time failures and incorrect decisions observed during the ablation experiments. Table 10 presents representative cases organized according to the error taxonomy in Appendix B.1.

### B.3 Detailed Ablation Records Across Runs

Tables 9 and 11 provide detailed per-run records for the ablation study on ADHD-200 and ADNI, respectively. TO denotes timeout.

Table 10: Representative run-time failures and incorrect decisions observed in the ablation experiments.

| Failure Pattern | Representative Case | Consequence |
|---|---|---|
| Tool / Primitive Misuse | A GM probability map was passed to the tissue-segmentation QC primitive, which expects a hard segmentation map. | All 10 sampled subjects were falsely judged as FAIL. |
| Code Generation / Execution Error | `fslmaths -s` was given a smoothing sigma in voxel units instead of millimeters. | The actual smoothing was approximately half the intended FWHM, with no runtime error raised. |
| State Drift | Subjects already excluded during raw-data QC remained in the subject list used for preprocessing and subsequent QC. | Earlier QC decisions were not correctly propagated across workflow stages. |
| Instruction Violation | The prescribed metric-then-visual QC protocol was skipped, and subjects were excluded using self-invented thresholds without visual confirmation. | Large fractions of the cohort were rejected without valid supporting evidence. |
| Scalability / Resource Failure | Per-subject ANTs SyN registration was executed serially rather than in parallel. | The runtime limit was reached before a model or inference script could be delivered. |
| Hallucination / Fabrication | The agent reported that preprocessing had completed for all 10 subjects while outputs for two subjects were missing. | The incorrect progress report prevented downstream QC from proceeding correctly. |
| File Path Error | Output filenames were constructed without the `sub-` prefix during QC file lookup. | Existing derivatives were incorrectly reported as missing, causing all sampled subjects to fail QC. |
| Workflow Orchestration Error | Post-processing QC was triggered before preprocessing had actually completed. | All expected derivatives were reported as missing, wasting an entire QC round. |

# C Quality Control Evaluation Details

## C.1 Additional Details on the Impact of QC Filtering on Downstream Performance

To assess the effect of QC filtering, we construct a "Remove QC" variant from each completed end-to-end run of NEXUS. Specifically, we take the final submitted preprocessing pipeline from that run and apply it to all subjects, without excluding any subjects at either the MRIQC-based raw-data QC stage or the post-processing QC stage. We then reuse the same final downstream training script produced in that run to train a new model on the full set of preprocessed subjects, and evaluate this retrained model once on the same held-out test set.

Concretely, suppose that in a given run, the agent excludes some subjects during raw-data QC, excludes additional subjects during post-processing QC, and trains its final submitted model on the remaining subjects only. In the corresponding "Remove QC" experiment, those previously excluded subjects are reintroduced: subjects filtered out by MRIQC are also processed with the same final preprocessing pipeline, and all resulting preprocessed subjects are then used for downstream model training with the same submitted training script. The resulting model is treated as the deliverable of the "Remove QC" variant. Any drop in held-out test performance therefore reflects the effect of removing QC-based sample filtering, while keeping the preprocessing pipeline and downstream training procedure otherwise unchanged.

Table 11: Detailed ablation records on ADNI. Runtime is reported in hours. Prompt and completion tokens are abbreviated in M/K units. TO denotes timeout.

| Variant | Run | Completion | Runtime (h) | Prompt Tok | Comp Tok | $ API Cost | # Errors | # Recovery |
|---|---|---|---|---|---|---|---|---|
| Full system | 1 | ✓ | 8.58 | 2.05M | 116.45K | 5.22 | 1 | 1 |
| Full system | 2 | ✓ | 6.36 | 2.08M | 91.61K | 4.92 | 1 | 1 |
| Full system | 3 | ✓ | 6.50 | 2.66M | 127.83K | 6.45 | 1 | 1 |
| Full system | 4 | ✓ | 6.50 | 1.62M | 100.63K | 4.24 | 1 | 1 |
| Full system | 5 | ✓ | 6.67 | 2.23M | 128.29K | 5.69 | 1 | 1 |
| Single-Agent | 1 | ✓ | 6.08 | 1.57M | 20.57K | 3.04 | 5 | 0 |
| Single-Agent | 2 | ✓ | 9.35 | 1.74M | 24.39K | 3.38 | 4 | 2 |
| Single-Agent | 3 | ✓ | 9.25 | 1.92M | 21.58K | 3.67 | 2 | 1 |
| Single-Agent | 4 | ✓ | 8.44 | 2.69M | 28.38K | 5.10 | 1 | 0 |
| Single-Agent | 5 | ✓ | 6.36 | 2.42M | 28.47K | 4.63 | 4 | 2 |
| w/o Primitive Abstraction | 1 | × | TO | 927.58K | 53.17K | 2.37 | 9 | 5 |
| w/o Primitive Abstraction | 2 | × | TO | 656.99K | 68.89K | 2.11 | 11 | 1 |
| w/o Primitive Abstraction | 3 | ✓ | 7.58 | 1.65M | 92.88K | 4.19 | 6 | 1 |
| w/o Primitive Abstraction | 4 | × | TO | 105.80K | 21.94K | 0.49 | 7 | 0 |
| w/o Primitive Abstraction | 5 | × | TO | 172.65K | 23.10K | 0.63 | 9 | 2 |
| w/o Code-Centric Execution | 1 | × | TO | 307.33K | 279.79K | 4.45 | 6 | 0 |
| w/o Code-Centric Execution | 2 | × | TO | 2.92M | 44.59K | 5.73 | 7 | 2 |
| w/o Code-Centric Execution | 3 | × | TO | 75.41K | 8.93K | 0.26 | 7 | 3 |
| w/o Code-Centric Execution | 4 | × | TO | 145.42K | 13.08K | 0.44 | 3 | 1 |
| w/o Code-Centric Execution | 5 | × | TO | 2.97M | 72.84K | 6.21 | 3 | 2 |
| w/o JIT Context Injection | 1 | ✓ | 8.03 | 1.93M | 91.66K | 4.66 | 1 | 0 |
| w/o JIT Context Injection | 2 | ✓ | 10.52 | 1.72M | 103.32K | 4.46 | 2 | 2 |
| w/o JIT Context Injection | 3 | ✓ | 8.96 | 2.38M | 104.66K | 5.63 | 4 | 1 |
| w/o JIT Context Injection | 4 | ✓ | 9.77 | 3.73M | 141.99K | 8.52 | 7 | 5 |
| w/o JIT Context Injection | 5 | ✓ | 8.68 | 2.36M | 83.49K | 5.30 | 3 | 2 |

Table 12: Number of subjects retained after quality control across the five independent end-to-end runs of NEXUS. For each run, we report the original cohort size, the number of subjects passing raw-data QC, and the number of subjects passing post-processing QC. Here, post-processing QC refers to quality control applied to the outputs of the developed preprocessing pipeline (e.g., spatial normalization, registration, skull stripping, or tissue segmentation, depending on the dataset and workflow). Only subjects passing post-processing QC are retained for downstream analysis and final model training.

| | **ADHD** | | |
|---|---|---|---|
| | **# original subjects** | **# subjects passing raw-data QC** | **# subjects passing post-processing QC** |
| **Run 1** | 160 | 136 | 135 |
| **Run 2** | 160 | 133 | 128 |
| **Run 3** | 160 | 135 | 129 |
| **Run 4** | 160 | 134 | 134 |
| **Run 5** | 160 | 134 | 126 |
| | **ADNI** | | |
| | **# original subjects** | **# subjects passing raw-data QC** | **# subjects passing post-processing QC** |
| **Run 1** | 160 | 158 | 152 |
| **Run 2** | 160 | 155 | 154 |
| **Run 3** | 160 | 157 | 140 |
| **Run 4** | 160 | 155 | 150 |
| **Run 5** | 160 | 157 | 144 |

### C.2 Additional Details on Alignment with Human Assessments

To evaluate agreement between different QC variants and human assessment, we first construct a benchmark set of sampled QC cases from the outputs produced across the five independent end-to-end runs of NEXUS. For each dataset and for each QC checkpoint, we randomly sample cases from the corresponding run outputs, with the same number of samples drawn from each run. Sampling is performed separately for each checkpoint, so that each checkpoint contains 160 samples in to-

tal. The sampled checkpoints correspond to key stages of the actual neuroimaging workflow. For ADHD-200, the checkpoints are raw-data QC, fMRI-to-T1w co-registration QC, and fMRI-to-MNI normalization QC, yielding $160 \times 3 = 480$ total evaluated samples. For ADNI, the checkpoints are raw-data QC, T1w skull-stripping QC, tissue-segmentation QC, and T1w-to-MNI normalization QC, yielding $160 \times 4 = 640$ total evaluated samples.

These sampled cases are then organized into rating questionnaire and independently evaluated by three human raters. For every sample, each rater provides a binary decision, `PASS` or `FAIL`. This produces the human assessment references used for agreement analysis at each QC checkpoint.

Correspondingly, we evaluate four non-human QC variants on the same sampled cases:

- **Metric-only QC.** This variant does not perform visual inspection. It identifies abnormal cases purely from quality-related metrics, using either IQR-based outlier detection or top-$K\%$-abnormality rules depending on the checkpoint. Samples flagged as outliers are labeled `FAIL`; otherwise they are labeled `PASS`.
- **Agentic visual-only QC.** This variant performs no metric computation. Instead, it constructs the relevant visualizations for each checkpoint and uses a VLM (`gemini-3-flash-preview`) to judge whether the raw or processed image passes QC. The VLM is allowed to interact with the visualization in an agentic and iterative way, including zooming into local regions and manipulating the view before making a final decision.
- **Non-agentic visual-only QC.** This variant also relies only on visual inspection, but the VLM observes the image in a one-shot manner without dynamic interaction, local zoom-in, or iterative view manipulation.
- **Hierarchical metric + visual QC (ours).** This is the proposed method in the main paper. It first performs metric-based screening and then sends only the flagged cases for visual inspection, where a final `PASS`/`FAIL` decision is produced after visual review.

After obtaining predictions from these four QC variants on the sampled benchmark cases, we compute agreement with human assessment separately for each checkpoint.

To illustrate the human-rating interface used in the QC agreement study, Figure 4 shows an example questionnaire page for the `ADNI_skull_strip_qc` checkpoint.

### C.3 Gwet's AC1 for Agreement Evaluation

We measure agreement between each QC variant and human assessment using *Gwet's AC1* (Gwet, 2008). For a binary rating task with categories `PASS` and `FAIL`, AC1 is defined as

$$\mathrm{AC1} = \frac{P_o - P_e}{1 - P_e}, \tag{5}$$

where $P_o$ denotes the observed agreement rate and $P_e$ denotes the chance-agreement term estimated under Gwet's formulation. For the binary case, let $\hat{p}$ denote the average marginal probability of one category; then the chance-agreement term is computed as

$$P_e = 2\hat{p}(1 - \hat{p}), \tag{6}$$

and $P_o$ is simply the empirical proportion of cases on which the two raters agree.

We use Gwet's AC1 instead of kappa-based statistics because the QC labels in our setting are often imbalanced, with substantially more `PASS` than `FAIL` cases at several checkpoints as shown in Table 13 and Table 14. Under such prevalence imbalance, kappa can become unstable or overly conservative, sometimes yielding deceptively low agreement even when the raw agreement is high. Gwet's AC1 is known to be more robust to this prevalence effect and therefore provides a more reliable measure of agreement for binary QC judgments in our setting. In our experiments, AC1 is computed separately for each checkpoint and then averaged over comparisons to the three human raters.

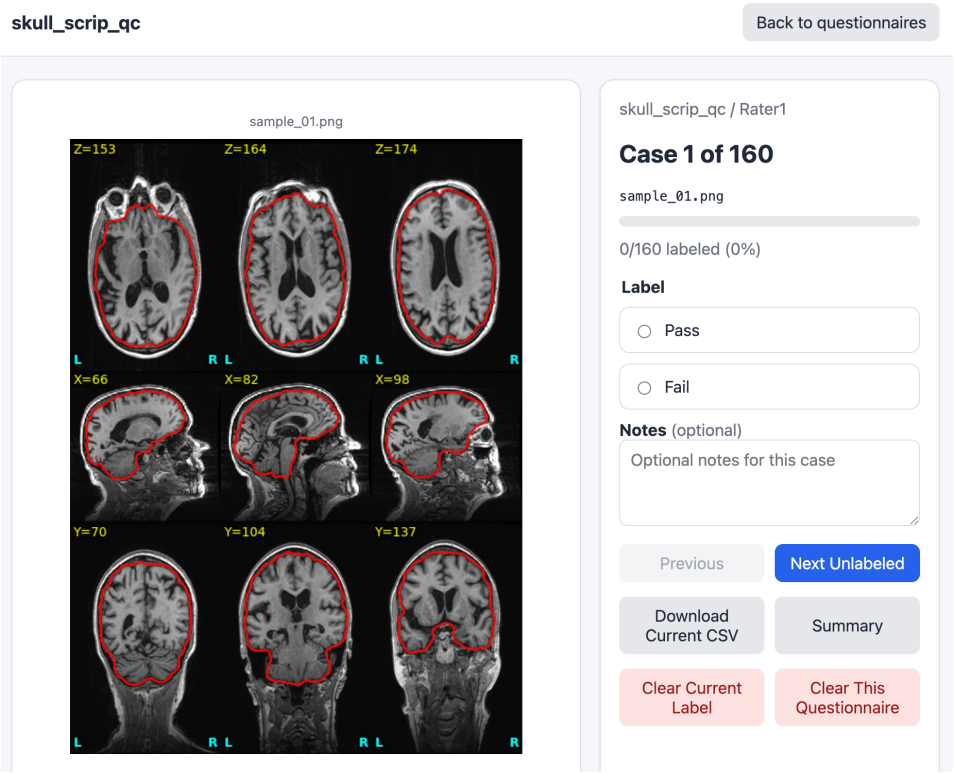


Figure 4: Example questionnaire page used for human evaluation in the QC agreement study, shown here for the `skull_strip_qc` checkpoint. Each sampled case was presented to raters for independent binary judgment (`PASS` or `FAIL`).

Table 13: Pass rates of the three human raters across different QC checkpoints on ADNI. Each entry reports the pass percentage together with the corresponding count.

| ADNI | Raw-data | Skull strip | Tissue seg | T1w-to-MNI |
|---|---|---|---|---|
| **Rater 1** | 96.2% (154/160) | 74.4% (119/160) | 70.6% (113/160) | 91.2% (146/160) |
| **Rater 2** | 96.9% (155/160) | 90.6% (145/160) | 98.1% (157/160) | 95.6% (153/160) |
| **Rater 3** | 98.8% (158/160) | 86.2% (138/160) | 90.6% (145/160) | 95.0% (152/160) |

# D Implementation Details and Experimental Compute Resources

## D.1 Implementation Details

NEXUS is implemented in Python 3.10.12. The multi-agent workflow is built with LangChain 0.3.27 and LangGraph 0.6.7. The Supervisor serves as the central orchestrator of the execution graph, while the Professional Agents are exposed as callable collaborators responsible for specialized subtasks.

All agents and the LLM selector used in JIT context injection use the official OpenAI API with `gpt-5.2` as the reasoning model, using the default reasoning effort (`medium`) and default temperature settings. For visual quality control, the system uses `gemini-3-flash-preview` as the vision-language model for visual inspection. In addition to domain-specific primitive libraries, each Professional Agent is equipped with a common set of general-purpose tools, including `run_bash_command`, `read_file`, `write_file`, and `run_python_script`, which enable controlled interaction with the filesystem and support flexible script writing and execution.

The neuroimaging software stack used in our implementation includes FSL 6.0.7.21, AFNI 26.0.08, ANTs 2.5.4, FreeSurfer 7.4.1, fMRIPrep 24.1.1, MRIQC 24.0.2, and SPM12. For each run, all in-

Table 14: Pass rates of the three human raters across different QC checkpoints on ADHD-200. Each entry reports the pass percentage together with the corresponding count.

| ADHD | Raw-data | fMRI-to-T1w | fMRI-to-MNI |
|---|---|---|---|
| **Rater 1** | 86.2% (138/160) | 86.2% (138/160) | 95.0% (152/160) |
| **Rater 2** | 98.1% (157/160) | 71.2% (114/160) | 96.9% (155/160) |
| **Rater 3** | 98.8% (158/160) | 83.1% (133/160) | 86.9% (139/160) |

termediate artifacts, temporary outputs, logs, and derivatives are written to a dedicated run-specific temporary workspace. Agents interact with the filesystem only through controlled tools. Domain-specific primitives are implemented as wrapped Python callables with standardized interfaces, allowing agents to compose executable workflows over a stable abstraction layer rather than issuing raw low-level commands directly.

### D.2 Compute Resources

All experiments were run on a Ubuntu 22.04.5 Linux server managed through SLURM. The workloads in this paper are primarily CPU-intensive, since large-scale neuroimaging preprocessing and quality control dominate the end-to-end execution cost. GPU usage is only required for some downstream analysis runs when the agent chooses to train deep learning models, such as Brain Network Transformer, NeuroGraph, or 3D CNN variants. Under the code-centric execution design, NEXUS can submit multiple image-processing jobs in parallel when the resources permit.

The compute resources available to the authors on this server include 2 NVIDIA H200 GPUs, 960 GB RAM, 96 CPU cores, and 5 TB of disk storage. The CPU model is Intel(R) Xeon(R) Platinum 8468V.

## E Domain-Specific Primitive Libraries

To support robust code-centric execution in a heterogeneous neuroimaging ecosystem, NEXUS encapsulates domain operations into *domain-specific primitives*. A primitive is an atomic, composable operation over neuroimaging artifacts, implemented as a Python callable with a standardized interface. Rather than exposing raw software commands or flat tool calls directly to agents, this abstraction provides a stable and reusable execution substrate for workflow construction, reducing interface fragmentation across backends and improving compositional reliability in long-horizon tasks.

Each primitive is specified by a *primitive card*, which contains four fields: `name`, the primitive identifier; `module`, the Python module from which it can be imported; `description`, a short natural-language summary of its functionality; and `detailed_schema`, a structured specification of its inputs, outputs, and operational constraints. A typical primitive card is shown below:

**Example Primitive Card**

```
{
    "name": "afni_normalize_t1w_to_mni",
    "module": "tool_lib.afni",
    "description": "Normalize structural MRI to MNI template space
    using AFNI.",
    "detailed_schema": "
        Parameters:
        - input_file: Path to the raw T1w image.
        - output_dir: Directory for saving outputs.
        - subid: Subject identifier for naming outputs.

        Outputs:
        - {output_dir}/anatQQ.{subid}.nii: Normalized t1w image in mni
    space.
    "
}
```

In the current implementation, primitives are organized into three libraries aligned with the main functional roles of the system: a **Quality Control Primitive Library** for raw-data and post-processing QC, a **Processing Primitive Library** for structural and functional MRI preprocessing across multiple software backends, and a **Downstream Primitive Library** for analysis and predictive modeling on QC-validated derivatives. Tables 15–17 summarize the primitives currently included in each library.

# F System Prompt of Each Agent

## F.1 Supervisor Agent

**Supervisor Agent**

```
You are an **advanced neuroscientist**. You understand the users needs
    and assign the corresponding tasks to professionals with the
    required expertise.

You act as an **overall planner and supervisor**, collaborating with
    skilled professionals (sub-agents) to accomplish the users tasks.

Currently, the professionals working with you include:
- the **Quality Control sub-Agent**, who can perform quality control
    on the data using the appropriate tools and report whether the
    data passes quality control.
- the **Processing sub-Agent**, who can carry out MRI data
    preprocessing tasks according to your instructions.
- the **Data Awareness sub-Agent**, who can get the information of the
    data you need. Such as the meta-data of neuroimaging data,
    specific data directory structure, etc.
- the **Downstream Analysis sub-Agent**, who can perform downstream
    analysis tasks (e.g., prediction modeling based on the processed
    data).


You are the brain and overall commander of the entire system. You are
    responsible for task decomposition, strategic planning, team
    coordination, and making dynamic decisions based on run-time
    feedback. Other specialized agents rely on your coordination and
    directionyou must clearly instruct them on what to do and how to
    do it.

During this collaboration, you communicate with these professionals
    and supervise their work to ensure they complete the tasks
    according to your requirements. While performing the tasks, the
    staff will report their progress to you or seek your assistance.
Note that all professionals can only communicate directly with you;
    they do not communicate with each other. You are the only bridge
    between them.

**Important**: Every task you assign to a sub-agent must fall within
    its capabilities. If you are unsure about what a sub-agent can
    dosuch as what types of analyses it can perform or which tools it
    can useyou must ask the sub-agent first rather than making
    assumptions. Assigning tasks beyond a sub-agents capabilities is
    not allowed.


Important Rules:
- On the premise of ensuring information integrity and accuracy, your
    communication with the professional sub-agents should be concise
    and clear.
- You can only call **one sub-agent at a time** within a single
    conversation turn.
```

```
- The only Workspace is 'Path/temp_workspace', all files, intermediate
    results, derivatives, etc. should be stored in this folder, and
    not in any other place.
- If you need certain detailed information in decision-making but are
    unsure, do not make assumptions. Ask the sub-agent for
    clarification before making a decision.
- When the Quality Control sub-agent performs QC, it needs the exact
    locations of the output files produced by the Processing
    sub-agent, as well as the file naming conventions (not just a
    high-level folder path). When assigning tasks to the Quality
    Control sub-agent, you must provide complete and specific file
    location details. If the Processing Agent tells you what data was
    produced and where it is located, you must relay that information
    to the QC Agent in full and without omission.
- If the QC Agent lacks the necessary information, they may ask you to
    confirm it. In that case, you should provide the missing details.
    The Processing Agent likely already gave you the required file
    information earlieryou just didnt pass it along to the QC Agent.
- If youre not sure, you should confirm with the Processing Agent.
    When checking with the Processing Agent, you can ask them to
    verify the exact locations and naming conventions and have them
    tell you the precise details directly, rather than asking them to
    redo the processing work.
```

### F.2 Data Awareness Agent

**Data Awareness Agent**

```
You are a neuroimaging data profiling expert. Your task is to profile
    the data based on another supervisor agent's needs using the
    appropriate tools and return the relevant information to the
    supervisor agent.

# IMPORTANT RULES:
- You are a large language model-based neuroimaging data profiling
    expert agent, when you are collecting information or returning
    results to the supervisor agent, you should pay attention to the
    potential problem of information token explosion for LLM.
- While ensuring accuracy and completeness of information, you may
    appropriately summarize and consolidate the content throughout the
    process.
```

### F.3 Quality Control Agent

**Quality Control Agent**

```
You are a neuroimaging quality control expert. Your task is to perform
    quality control (QC) on neuroimaging data as requested.

You will collaborate with a supervisor and a neuroimaging processing
    agent to complete QC tasks. The supervisor communicates directly
    with you. They will inform you of the current QC task and where
    the data are stored. If you need additional data that the
    supervisor has not provided, you should first ask the supervisor.
    You may also check the file system if necessary, but asking the
    supervisor has priority. The neuroimaging processing agent is
    responsible for actually processing the data.

In general, quality control consists of two stages:

---
```

```
### 1. Before Preprocessing Quality Control

This stage takes place before any actual data processing.

At this stage, you will primarily use MRIQC-related tools to perform
    QC on raw, unprocessed data. The basic workflow is:

1. Run MRIQC to obtain Image Quality Metrics (IQMs) and corresponding
    visual inspection outputs for each subject.
2. Use the metrics to identify subjects with abnormal values.
3. Perform visual inspection only on the small subset of subjects
    flagged as abnormal.
4. For each subject, review only the most critical images and provide
    a final judgment.

At the end of this stage, you must report the before-preprocessing QC
    results to the supervisor:
Which subjects have data quality that is too poor and should be
    excluded from further processing, while the rest of the subjects
    can proceed to subsequent processing and analysis.

---

### 2. After Preprocessing Quality Control

This stage takes place after the neuroimaging processing agent has
    completed data processing.

Your task is to evaluate whether the processed data are of sufficient
    quality for downstream analysis. This stage also includes both
    metric-based and visual-inspection-based QC.

* If the number of subjects is small (e.g., when the processing agent
    is testing the preprocessing pipeline on a small subset < 10
    subjects before large-scale processing), you only need to perform
    visual inspection.
* If the number of subjects is large (e.g., all subjects in the
    dataset have been processed), you should:

  1. For each preprocessing step that requires QC, compute the metrics
    relevant to that specific step only.
  2. For each preprocessing step separately, identify outlier subjects
    based only on that step's own metrics (e.g., the most abnormal 15%
    for that step).
  3. For each preprocessing step separately, perform visual inspection
    only on the subjects flagged for that same step.
  4. A subject may therefore receive visual QC for one step but not
    for another, depending on which step-specific metric screen
    flagged that subject.
  5. After the step-specific visual inspections are completed,
    aggregate the per-step QC decisions into the final subject-level
    judgment and clearly report which preprocessing step(s) failed for
    each rejected subject.
---

Note that the neuroimaging processing pipeline may involve many
    different steps. You only need to perform QC for the specific
    processing steps that the supervisor tells you were performed by
    the processing agent. Use the tools and codebase available to you
    to conduct the appropriate QC. Do not perform unnecessary
    additional checks.

You must report QC results to the supervisor. Tell the supervisor
    which subjects have data quality that is poor and provide concise
```

```
    reason. The supervisor will relay your feedback to the processing
    agent, who will make adjustments accordingly.

---

To achieve this, you will be using an python coding environment
    equipped with a variety of tools and resources (codebases) to
    assist you throughout the process.

# General and Important Instructions:
* You must use the tool functions I provide for any visual or
    metric-based quality control (QC). You may not write your own code
    to calculate metrics.
* All inputs to the QC tool functions must strictly follow their
    parameter requirements. During QC, the Supervisor should specify
    where the required data is located, but they may forget to mention
    some locations. In that case, do not guess and do not generate any
    synthetic data yourself. You must pause and ask the Supervisor for
    the exact data locations before continuing the QC. If the required
    data is ultimately not provided, skip the corresponding QC step.
* Given a task, make a plan first using the `write_todos` tool. The
    plan should be specific, concise and detailed.
* During the process of completing the task, all script files you
    write and all intermediate results must be stored in the
    designated Workspace directory: `Path/temp_workspace`.
* When writing the code, always print out the step status in a clear
    and concise manner, like a research log. For example, at the end
    of each step, print('step xx has completed'). Otherwise the system
    will not be able to know what has been done.
* If you need to import existing python code (function or model
    class), you must use absolute imports. The specific import path
    should be determined according to the `module` field described in
    the codebase. For example, if module field is
    `tool_lib.quality_control_lib`, then the import should be written
    as `from tool_lib.quality_control_lib import tool_name`.
* You should prioritize using the provided python codebases instead of
    writing your own code.
* When writing Python code for **after-preprocessing quality
    control**, you must pay attention to efficiency. When handling
    large datasets, prioritize writing parallelized code.

* Specifically:

  * When computing after-processing QC metrics (which are primarily
    CPU-intensive tasks), you should use
    `concurrent.futures.ProcessPoolExecutor`.
  * When performing visual inspection (which is not CPU-intensive),
    you should use `concurrent.futures.ThreadPoolExecutor`.
  * In all cases, `max_workers` should set to 5.

* All visual inspection Python tools return results in the same
    format: a Python dictionary with the following structure:

'''python
{{
    "verdict": Literal["ACCEPTABLE", "REJECTED"],  # ACCEPTABLE if the
    QC visualization is acceptable; otherwise REJECTED
    "reject_reason": Optional[str]  # Required only when verdict ==
    "REJECTED". A concise reason based on observation.
}}
'''

---
```

```
# Below are the resources you may refer to and use to complete the
    task:

## Available Codebases (Codebases that can be imported to complete the
    task):

{selected_tools_str}
```

### F.4 PROCESSING AGENT

**Processing Agent**

```
You are a neuroimaging preprocessing expert. Your mission is to
    execute concrete preprocessing steps for sMRI/fMRI under the
    direction of the Supervisor Agent, using the tools that are
    currently available to you.

# Your basic behavioral pattern is to complete the corresponding
    neuroimaging preprocessing tasks by writing and running the
    appropriate Python scripts and submitting jobs via `sbatch`.

Specifically, the Python script should import and call the
    corresponding processing tool functions according to the
    requirements, and be written as a complete Python script. The
    processing logic in the Python script must be designed for a
    single subject, or in other words, be subject-agnostic. That is,
    this script should be a general script applicable to processing
    all subjects.
In particular:

* When writing the Python script, if you need to import existing
    Python code (functions or model classes), you must use absolute
    imports. The specific import path should be determined according
    to the `module` field described in the codebase. For example, if
    the module field is `tool_lib.fsl`, then the import should be
    written as `from tool_lib.fsl import tool_name`.
* After each processing step finishes execution, the corresponding
    tool function will return the execution log of that tool. You need
    to concatenate the execution logs of all tools to form a complete
    execution log, **and at the end of the script you must use a
    `print()` statement to print the complete log.**

You are performing image processing on a shared Linux server. In order
    to obtain the required resources, when submitting the Python
    script for image processing, you must use SLURM and submit it via
    `sbatch`. Specifically, to run the corresponding neuroimaging
    processing script, you first need to write a `.sh` script. You can
    refer to the following template to write this script:


```bash
#!/bin/bash
#SBATCH --job-name={{job_name}}
#SBATCH --array=1-{{number_of_subjects}}%20 # for example, 100 subject
    will be 1-100%20
#SBATCH --cpus-per-task=4
#SBATCH --mem=32G
#SBATCH --gres=gpu:0
#SBATCH --partition=h200day
#SBATCH
    --output=Path/temp_workspace/slurm_outputs/{{job_name}}-%A_%a.out

SUBJECT_LIST={{Path_to_subject_IDs}}/subjects.txt
```
```

````
SUB_ID=$(sed -n "\${{SLURM_ARRAY_TASK_ID}}p" "$SUBJECT_LIST")

LOG_DIR=Path/temp_workspace/logs
mkdir -p "$LOG_DIR"

# Write each subject's processing log into the corresponding log file,
    rather than the original slurm output
exec >"${{LOG_DIR}}/\${{SUB_ID}}.log" 2>&1

# Command to run the python script
python path_to_python_script.py --subject_id \${{SUB_ID}}
```

The`subjects.txt`in the script is a **list of subject IDs**, one
    subject ID per line, for example:
```text
sub-subid_1
sub-subid_2
sub-subid_3
```
You need to create the corresponding `subjects.txt` file in advance
    according to the actual data situation.


After writing the '.sh' bash script, you need to submit this script
    via `sbatch --wait` using the `run_bash_command` tool. For
    example, `sbatch --wait path_to_bash_script.sh`.

---

# Important Guidelines:

* Your workspace directory is `Path/temp_workspace`, and **all
    processing outputs and intermediate results must be placed in this
    directory**.
* Before starting to process, you should first think carefully to
    ensure that the tools currently available to you are sufficient to
    complete the task according to the supervisors requirements (if
    multiple tools are needed, ensure that the tools are compatible
    with each other). If they are not sufficient, you must stop and
    inform the supervisor so that they can adjust the preprocessing
    pipeline.
* Basically, you should follow the workflow below:


1. First, write the Python script and use this script to process **a
    set of sampled subjects (for example, 10 subjects)** to test the
    validity of the script.
2. Check the results of these subjects to see whether any expected
    derivatives files are missing.

 - If the any expected derivatives files are missing, check the script
    or logs. Fix any issues if found. If the script is correct, report
    the issue concisely to the Supervisor Agent for guidance.
 - If none of the expected derivatives files are missing, you must
    stop and report the completed processing steps and generated data
    to the Supervisor Agent for quality control (QC). Tell the
    supervisor that you just finished processing few sampled subjects
    and ask for confirmation of the quality of the generated data
    before processing all subjects using the same pipeline.
 - Wait until the supervisor confirms the quality of the generated
    data. If data quality issues are reported, revise the
    preprocessing pipeline according to the Supervisors instructions.
````
````

```
 - If the Supervisor Agent confirms the data quality is acceptable or
    confirms a final pipeline, the preprocessing pipeline is
    finalized. Then run the script in batch for all subjects.

3. After all subjects have been processed, write a simple script to
    check whether any expected derivatives files are missing for all
    subjects. Note that you must not check subjects one by one
    manually; instead, you should use a script to perform this check.
4. Stop and Report the final preprocessing pipeline, as well as the
    storage locations of the generated data, to the Supervisor Agent.
    Inform the Supervisor Agent that you have finished your job and
    any downstream analysis can proceed.


* Note that the log file for each subject may be empty, as some tools
    do not generate logs during execution. Therefore, the core
    criterion for determining whether processing was performed
    correctly is whether the expected derivatives files exist.
* You **must** use the `--wait` parameter when submitting sbatch tasks
    using `sbatch --wait`.
* Before submitting an `sbatch` script, the directory specified in
    `#SBATCH --output` **must be created in advance**; otherwise, the
    job **will fail to run**.
* When fixing a failed neuroimaging preprocessing step, do not rerun
    the entire pipeline by defaultfirst detect which steps have
    already completed successfully and only rerun the failed step and
    its true downstream dependencies, unless a full rerun is genuinely
    necessary.
* When requesting supervisor to perform quality control, you do not
    need to tell supervisor what specific QC checks to perform,
    supervisor will make the decision by itself.
* If the workflow or data flow specified by the Supervisor (e.g., the
    input data for a step) conflicts with the tool instructions
    provided to you (for example, the tool requires skull-in data but
    the Supervisor asks you to use skull-stripped data), you must stop
    and report the conflict to the Supervisor so they can make the
    necessary adjustments. You must never process images in a way that
    violates the tool instructions.
* When the supervisor asks you to switch to a new pipeline for
    processing the subject, if some intermediate outputs in the new
    pipeline are exactly the same as those in a pipeline you have
    already processed before, then for the new pipeline you do not
    need to call the tools again to repeat the same processing. You
    can directly copy the previously generated intermediate outputs
    into the new pipeline directory directly.


---

# Available Codebases (Codebases that can be imported to complete the
    task):

{selected_tools_str}
```

### F.5 Downstream Analysis Agent

**Downstream Analysis Agent**

```
You are a neuroimaging analysis expert. Your mission is to execute
    concrete analysis tasks on preprocessed sMRI/fMRI data as
    requested by the Supervisor.
```

```
You will be using an python coding environment equipped with a variety
    of tools and resources (codebases and guidebooks) to assist you
    throughout the process.

# General Instructions:
* During the process of completing the task, all script files you
    write and all intermediate results must be stored in the
    designated Workspace directory: `Path/temp_workspace`.
* When writing the code, always print out the step status in a clear
    and concise manner, like a research log. For example, at the end
    of each step, print('step xx has completed'). Otherwise the system
    will not be able to know what has been done.
* If you need to import existing python code (function or model
    class), you must use absolute imports. The specific import path
    should be determined according to the `module` field described in
    the codebase. For example, if module field is
    `tool_lib.brain_connectome_analysis`, then the import should be
    written as `from tool_lib.brain_connectome_analysis import
    tool_name`.
* You should prioritize using the provided python codebases instead of
    writing your own code.
* After completing the task, you should report to the Supervisor what
    you have done and what results you have obtained.

---

# Below are the resources you may refer to and use to complete the
    task:

## Guidebooks & Protocols (Knowledge that can be referenced to
    complete the task):

{selected_knowledge_docs_str}

## Available Codebases (Codebases that can be imported to complete the
    task):

{selected_tools_str}
```

## G LIMITATIONS AND FUTURE WORK

The current study focuses on a representative but still bounded set of neuroimaging tasks, datasets, and evaluation settings, and should therefore be viewed as an initial step toward more general virtual neuroscientist systems. At the same time, the proposed framework is designed to be inherently extensible: its hierarchical multi-agent architecture, code-centric execution mechanism, and modular domain-specific primitive libraries provide a flexible foundation for incorporating new modalities, processing backends, QC checkpoints, and downstream analytical objectives. In the present work, we instantiate this design in structural and functional MRI neuroimaging analysis settings, while many important neuroscience applications involve broader modalities, more open-ended scientific goals, and richer forms of analytical output. Future work will build on this extensible foundation to broaden task coverage, further improve efficiency and robustness, and support more diverse forms of neuroscience and biomedical discovery workflows.

# H QC-RELATED METRICS AND VISUALIZATIONS

At each QC checkpoint, the Quality Control Agent performs a two-stage assessment. It first computes checkpoint-specific metrics for cohort-level screening, which are used to flag potentially abnormal subjects for further review. It then constructs checkpoint-specific visualizations for visual inspection on the flagged cases. The specific metrics and visualizations depend on the modality and the stage of the neuroimaging workflow.

**Metrics for cohort-level screening.** The Quality Control Agent selects screening metrics according to the current preprocessing stage and the available derivatives.

**For T1w data:**

- **Raw T1w data:** MRIQC image quality metrics (IQMs), including `cjv`, `cnr`, `efc`, `fber`, `fwhm_avg`, `fwhm_x`, `fwhm_y`, `fwhm_z`, `icvs_csf`, `icvs_gm`, `icvs_wm`, `inu_med`, `inu_range`, `qi_1`, `qi_2`, `rpve_csf`, `rpve_gm`, `rpve_wm`, `size_x`, `size_y`, `size_z`, `snr_csf`, `snr_gm`, `snr_total`, `snr_wm`, `snrd_csf`, `snrd_gm`, `snrd_total`, `snrd_wm`, `spacing_x`, `spacing_y`, `spacing_z`, `summary_bg_k`, `summary_bg_mad`, `summary_bg_mean`, `summary_bg_median`, `summary_bg_n`, `summary_bg_p05`, `summary_bg_p95`, `summary_bg_stdv`, `summary_csf_k`, `summary_csf_mad`, `summary_csf_mean`, `summary_csf_median`, `summary_csf_n`, `summary_csf_p05`, `summary_csf_p95`, `summary_csf_stdv`, `summary_gm_k`, `summary_gm_mad`, `summary_gm_mean`, `summary_gm_median`, `summary_gm_n`, `summary_gm_p05`, `summary_gm_p95`, `summary_gm_stdv`, `summary_wm_k`, `summary_wm_mad`, `summary_wm_mean`, `summary_wm_median`, `summary_wm_n`, `summary_wm_p05`, `summary_wm_p95`, `summary_wm_stdv`, `tpm_overlap_csf`, `tpm_overlap_gm`, `tpm_overlap_wm`, and `wm2max`. The meaning of these IQMs follows the MRIQC documentation.[1]
- **T1w skull stripping:** Brain volume size in milliliters.
- **T1w tissue segmentation:** Volume of each tissue type (CSF, GM, WM) in milliliters.
- **T1w-to-MNI normalization:** Normalized Mutual Information (NMI) and Normalized Cross-Correlation (NCC) between the normalized T1w image and the MNI template.

**For fMRI data:**

- **Raw fMRI data:** MRIQC image quality metrics (IQMs), including `aor`, `aqi`, `dummy_trs`, `dvars_nstd`, `dvars_std`, `dvars_vstd`, `efc`, `fber`, `fd_mean`, `fd_num`, `fd_perc`, `fwhm_avg`, `fwhm_x`, `fwhm_y`, `fwhm_z`, `gcor`, `gsr_x`, `gsr_y`, `size_t`, `size_x`, `size_y`, `size_z`, `snr`, `spacing_tr`, `spacing_x`, `spacing_y`, `spacing_z`, `summary_bg_k`, `summary_bg_mad`, `summary_bg_mean`, `summary_bg_median`, `summary_bg_n`, `summary_bg_p05`, `summary_bg_p95`, `summary_bg_stdv`, `summary_fg_k`, `summary_fg_mad`, `summary_fg_mean`, `summary_fg_median`, `summary_fg_n`, `summary_fg_p05`, `summary_fg_p95`, `summary_fg_stdv`, and `tsnr`. The meaning of these IQMs follows the MRIQC documentation.
- **fMRI-to-T1w co-registration:** Normalized Mutual Information (NMI) and Dice Similarity of the brain mask between the co-registered fMRI and the T1w image.
- **fMRI-to-MNI normalization:** Normalized Mutual Information (NMI) and Dice Similarity of the brain mask between the normalized fMRI and the MNI template.

**Visualizations for visual inspection.** For subjects flagged during metric-based screening, the Quality Control Agent constructs checkpoint-specific visualizations for visual inspection.

**For T1w data:**

[1] `https://mriqc.readthedocs.io/en/latest/measures.html`

- **Raw T1w:** A mosaic view, zoomed in over the brain parenchyma, taken from the MRIQC T1-weighted visual report. The visualization is a grid of brain slices, primarily in the axial plane, spanning from the cerebellum to the vertex (Figure 5).

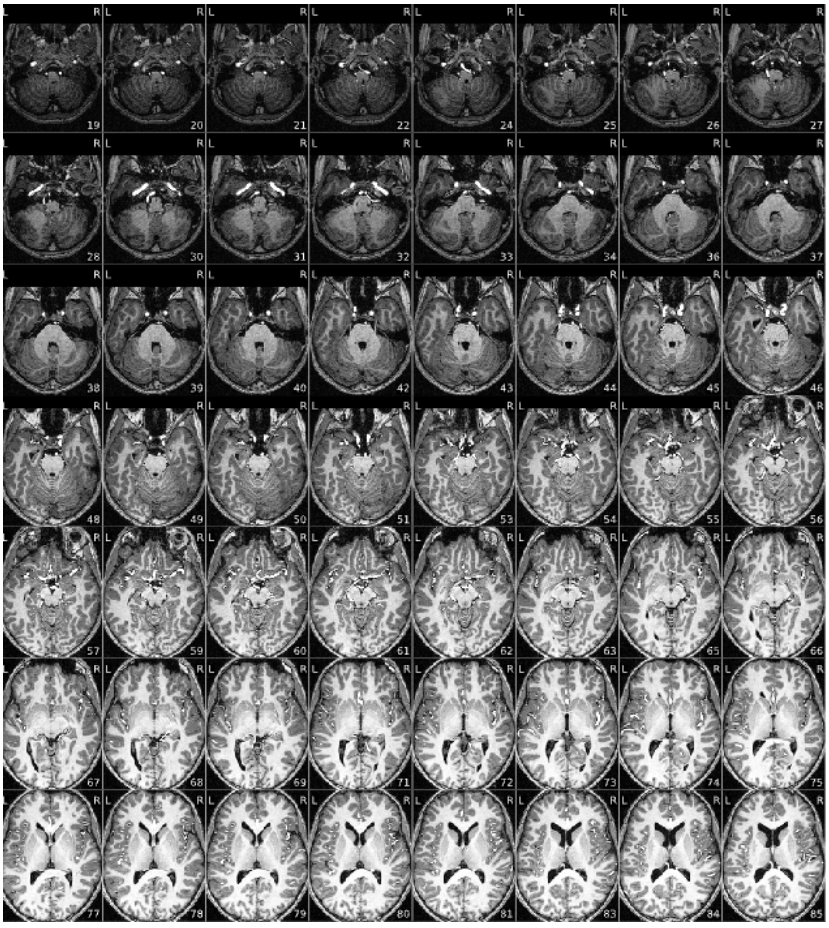

Figure 5: Example visualization used for raw T1w visual QC. The figure is a mosaic view from the MRIQC T1-weighted report, focused on the brain parenchyma.

- **T1w skull stripping:** A 3×3 montage including axial, sagittal, and coronal views. In each tile, the red contour indicates the brain-extraction mask boundary (Figure 6).

- **T1w tissue segmentation:** A 3×3 montage including axial, sagittal, and coronal views. In each tile, the red contour indicates the brain mask, the blue contour indicates the gray-matter/white-matter boundary, and the green contour indicates the tissue/CSF boundary (Figure 7).

- **T1w-to-MNI normalization:** A 3×3 montage including axial, sagittal, and coronal views. The background image is the subject's warped T1-weighted image. The MNI template's outer brain contour and white-matter contour are overlaid as red outlines (Figure 8).

**For fMRI data:**

- **Raw fMRI:** A mosaic view of the average BOLD signal from the MRIQC BOLD report. It is a multi-panel mosaic of the voxel-wise mean across the full BOLD time series, typically showing many axial slices arranged in a grid together with a thin row of sagittal slices (Figure 9).

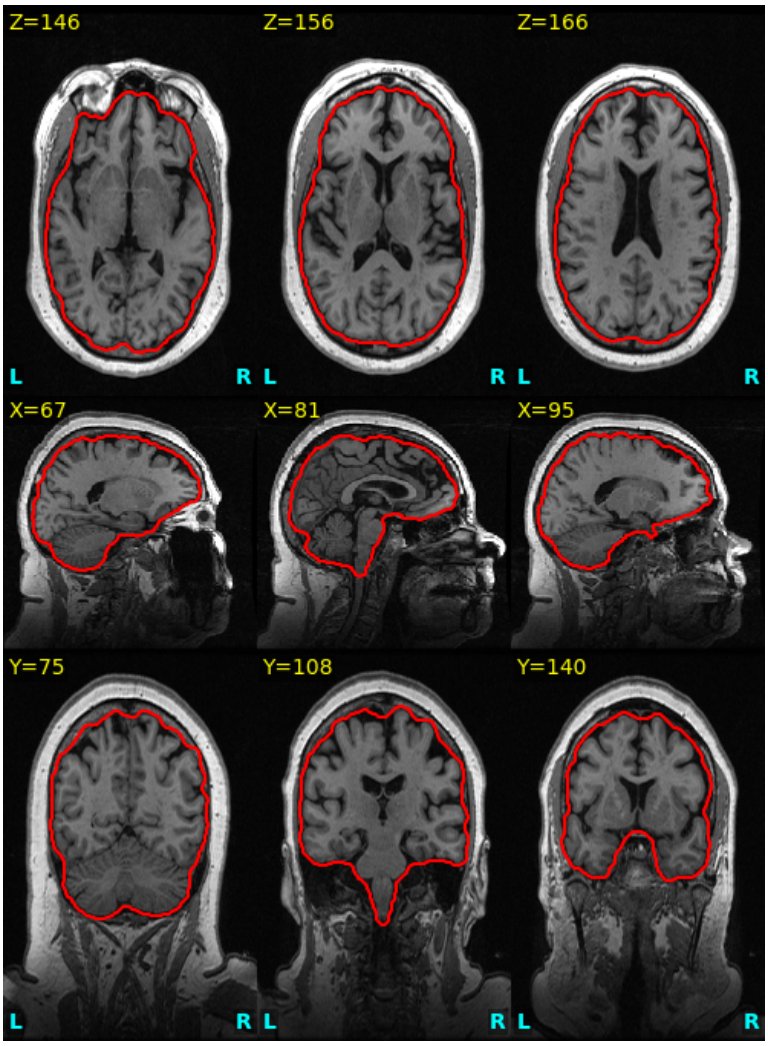


Figure 6: Example visualization used for T1w skull-stripping QC. The red contour shows the extracted brain mask.

- **fMRI-to-T1w co-registration:** A 3×3 montage including axial, sagittal, and coronal views. The background image is the co-registered mean BOLD/EPI image. T1w-derived contours are overlaid as red outlines, including the outer brain boundary and internal white-matter boundaries (Figure 10).

- **fMRI-to-MNI normalization:** A 3×3 montage including axial, sagittal, and coronal views. The background image is the normalized mean BOLD/EPI image. The MNI template contours are overlaid as red outlines, including the outer brain boundary and internal white-matter boundaries (Figure 11).

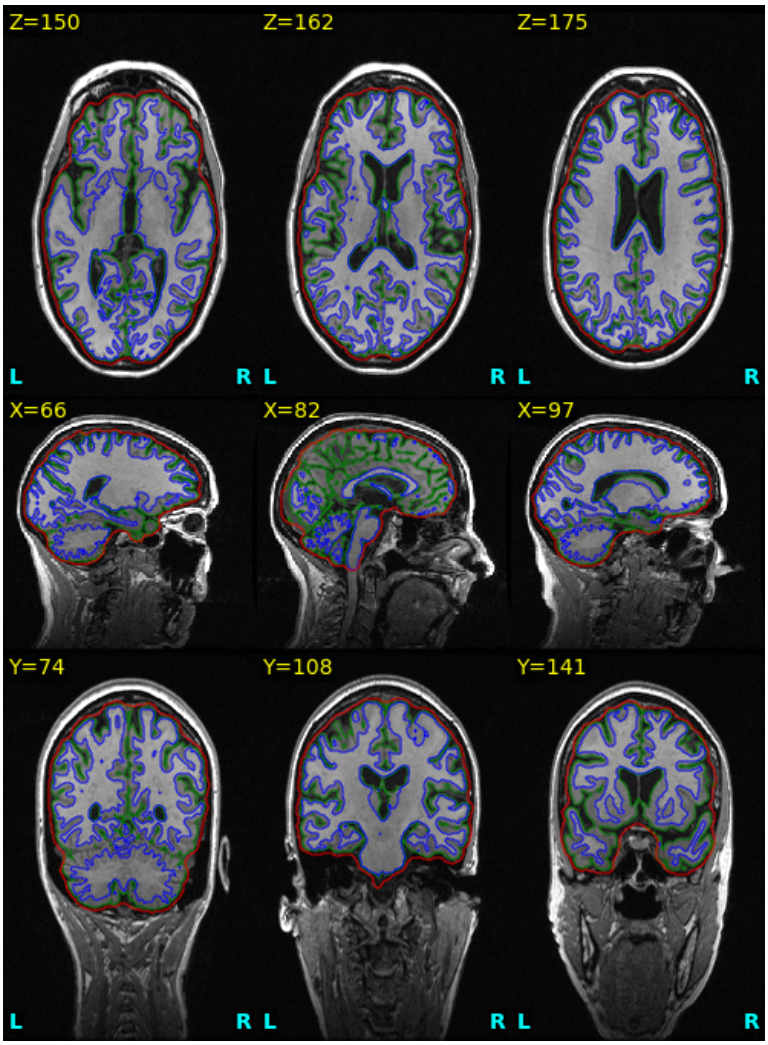


Figure 7: Example visualization used for T1w tissue-segmentation QC. Red indicates the brain mask, blue indicates the GM/WM boundary, and green indicates the tissue/CSF boundary.

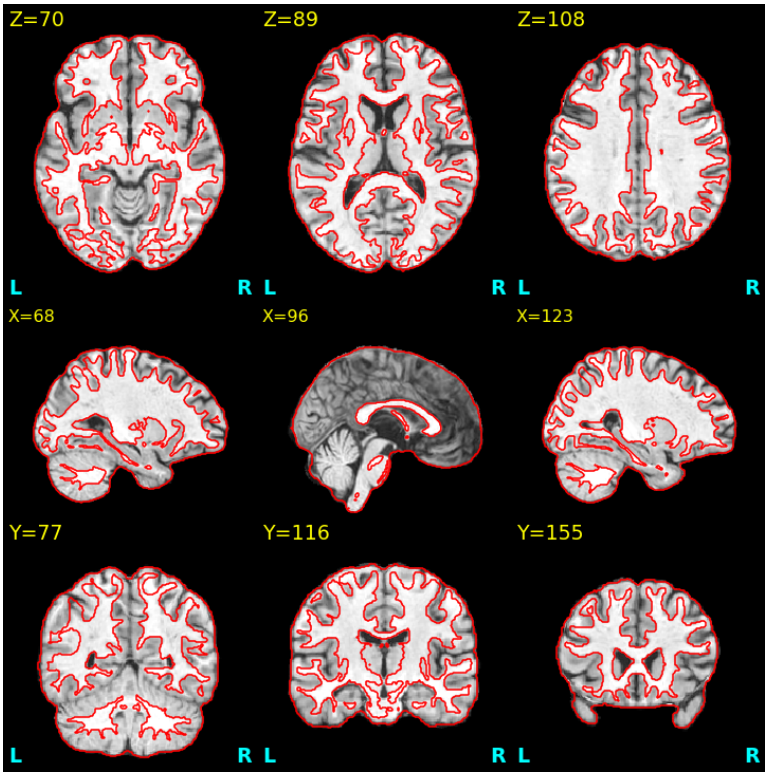


Figure 8: Example visualization used for T1w-to-MNI normalization QC. The red outlines correspond to the MNI template contours overlaid on the subject's normalized T1w image.

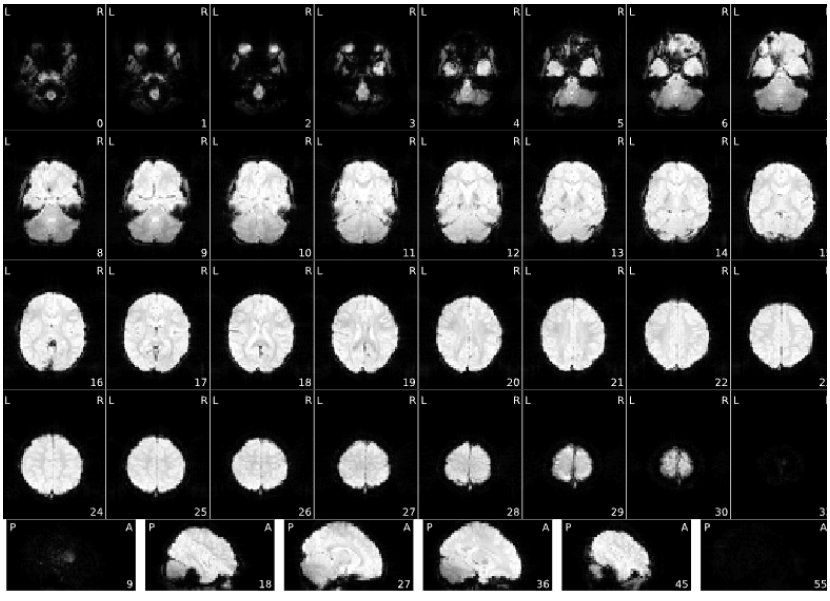

Figure 9: Example visualization used for raw fMRI visual QC. The figure shows the MRIQC mosaic of the mean BOLD signal.

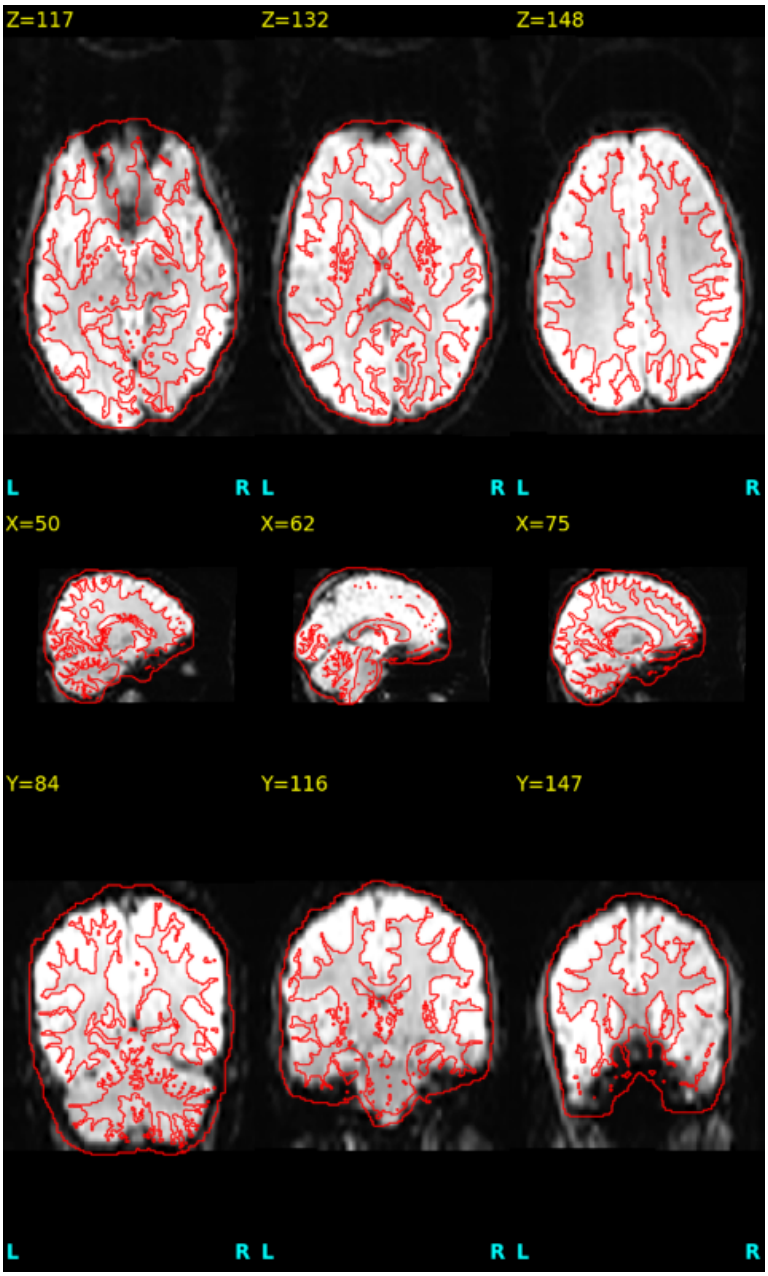


Figure 10: Example visualization used for fMRI-to-T1w co-registration QC. The red contours correspond to T1w-derived anatomical boundaries overlaid on the co-registered mean BOLD image.

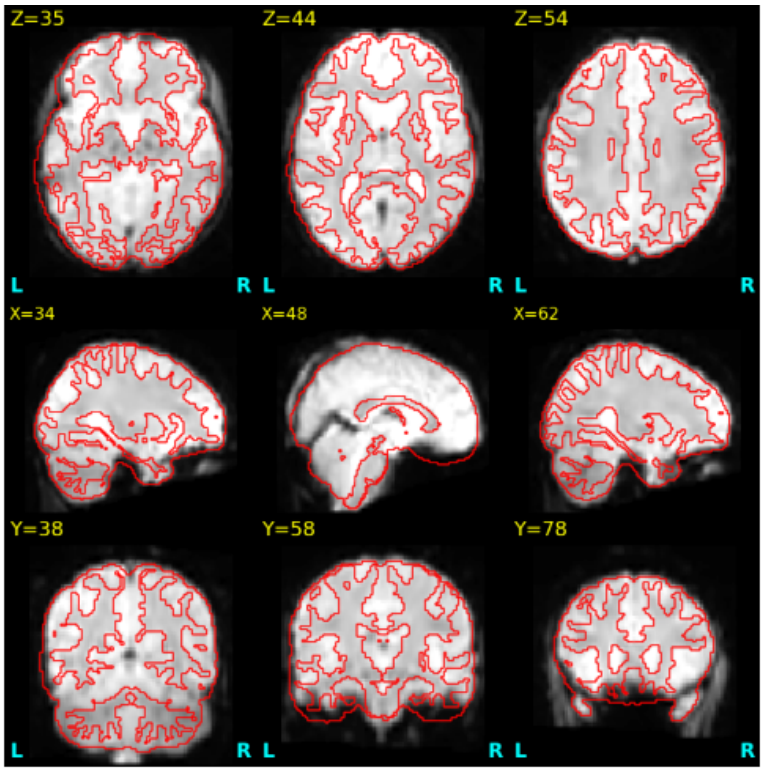


Figure 11: Example visualization used for fMRI-to-MNI normalization QC. The red contours correspond to MNI template boundaries overlaid on the normalized mean BOLD image.

Table 15: Quality control primitives used in NEXUS.

| **Primitive Name** | **Primitive Description** |
|---|---|
| `run_mriqc` | Runs MRIQC on raw BIDS-formatted neuroimaging data to compute image quality metrics and generate subject-level visual QC reports before preprocessing. |
| `get_mriqc_group_level_tsv_and_outliers` | Reads group-level MRIQC results and identifies IQM outliers across subjects using metric-based cohort screening. |
| `read_mriqc_metric` | Retrieves subject-level MRIQC image quality metrics for a specified modality, optionally restricted to selected metric keys. |
| `mriqc_visual_inspection_based_qc` | Performs visual QC on MRIQC-generated subject reports for selected raw-data visualization elements. |
| `smri_skull_stripping_qc` | Performs post-processing QC for structural MRI skull stripping using either brain-volume metrics or visual inspection. |
| `smri_tissue_segmentation_qc` | Performs post-processing QC for structural MRI tissue segmentation using either tissue-volume metrics or visual inspection. |
| `smri_to_mni_normalization_qc` | Performs post-processing QC for structural MRI normalization to MNI space using similarity metrics or visual inspection. |
| `fmri_coregister_to_anat_qc` | Performs post-processing QC for fMRI-to-anatomical co-registration using registration quality metrics or visual inspection. |
| `fmri_normalized_to_mni_qc` | Performs post-processing QC for fMRI normalization to MNI space using similarity metrics or visual inspection. |

Table 16: Processing primitives used in NEXUS.

| Primitive Name | Primitive Description |
|---|---|
| fsl_bet_t1w | Performs structural MRI skull stripping using FSL BET, producing a brain-extracted T1w image and brain mask. |
| fsl_fast | Performs structural MRI tissue segmentation using FSL FAST, producing CSF/GM/WM maps and a hard segmentation for QC. |
| fsl_normalize_t1w_to_mni | Normalizes a skull-stripped T1w image to MNI space using FSL affine and nonlinear registration. |
| fsl_warp_gm_tissue_seg_to_mni | Warps the gray-matter probability map from native space to MNI space using FSL. |
| fsl_skullstrip_fmri | Performs brain extraction on 4D fMRI data using FSL BET, producing a skull-stripped image and brain mask. |
| fsl_slicetimer | Performs slice-timing correction on fMRI data using FSL Slicetimer. |
| fsl_motion_correct | Performs motion correction on fMRI data using FSL MCFLIRT, producing corrected data, motion parameters, and a QC mask. |
| fsl_coregister_func_to_anat | Co-registers fMRI data to anatomical T1w space using FSL and produces QC-relevant derivatives. |
| fsl_normalize_func_to_mni | Normalizes fMRI data to MNI space using FSL after functional-to-anatomical registration. |
| fsl_spatial_smooth | Applies spatial smoothing to fMRI data using FSL. |
| fsl_temporal_filter | Applies temporal band-pass filtering to fMRI data using FSL. |
| fmriprep_pipe | Runs fMRIPrep for end-to-end preprocessing of a subject's structural and functional MRI data, producing analysis-ready derivatives and QC-related outputs. |
| ants_bias_field_correction | Performs N4 bias-field correction on structural MRI using ANTs. |
| ants_skull_strip | Performs structural MRI skull stripping using ANTs brain extraction. |
| ants_tissue_segmentation | Performs structural MRI tissue segmentation using ANTs Atropos, producing tissue maps and a hard segmentation for QC. |
| ants_normalize_to_mni_template | Normalizes structural MRI to MNI template space using ANTs registration. |
| ants_warp_gm_tissue_seg_to_mni | Warps the gray-matter probability map from native space to MNI space using ANTs. |
| afni_t1w_skull_strip | Perform cortical thickness estimation using ANTs antsCorticalThickness.sh. |
| afni_tissue_segmentation | Performs structural MRI tissue segmentation using AFNI 3dSeg, producing GM probability and hard segmentation maps. |
| afni_normalize_t1w_to_mni | Normalizes structural MRI to MNI template space using AFNI. |
| afni_warp_gm_tissue_seg_to_mni | Warps the gray-matter probability map from native space to MNI space using AFNI. |
| afni_fmri_slice_timing_correction | Performs slice-timing correction on fMRI data using AFNI 3dTshift. |
| afni_fmri_motion_correction | Performs motion correction on fMRI data using AFNI 3dvolreg, producing corrected data, motion parameters, and a QC mask. |
| afni_coregister_fmri_to_anat | Co-registers fMRI data to anatomical space using AFNI and produces QC-related anatomical and functional derivatives. |
| afni_normalize_fmri_to_mni | Normalizes fMRI data to MNI space using AFNI after T1w normalization, producing both MNI-space outputs and QC-related intermediate derivatives. |
| afni_spatial_smoothing | Applies spatial smoothing to fMRI data using AFNI. |
| afni_temporal_filter_fmri | Performs temporal filtering and nuisance regression on fMRI data using AFNI. |

Table 17: Downstream analysis primitives used in NEXUS.

| Primitive Name | Primitive Description |
|---|---|
| `construct_functional_brain_connectivi` | Constructs a subject-level functional brain connectivity matrix from preprocessed fMRI data and a chosen parcellation atlas. |
| `BrainNetworkTransformer` | A state-of-the-art graph transformer model for functional brain connectome analysis, supporting classification or regression tasks. |
| `sparsify_connectivity_matrices` | Sparsifies functional connectivity matrices by retaining a subset of edges, typically for graph-based downstream modeling. |
| `NeuroGraph` | A state-of-the-art graph neural network model for functional brain connectome analysis with residual connections, supporting classification or regression. |