DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

ORNL/TM-2025/4360

ESAC v1: Enhanced AI-Powered Chatbot for EQ-SANS Experiment Automation Improvements and Updates

Do, Changwoo
Nagy, Gergely
Heller, William T.

December 2025

OAK RIDGE National Laboratory

ORNL IS MANAGED BY UT-BATTELLE LLC FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2025/4360

Neutron Sciences Directorate

ESAC V1: ENHANCED AI-POWERED CHATBOT FOR EQ-SANS EXPERIMENT AUTOMATION IMPROVEMENTS AND UPDATES

Do, Changwoo
Nagy, Gergely
Heller, William T.

December 2025

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831
managed by
UT-BATTELLE LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

FOREWORD

This technical memorandum documents the significant improvements and updates made to the ESAC (EQ-SANS Assisting Chatbot) system since its initial publication in SoftwareX journal. ESAC v1 represents a major advancement in AI-assisted neutron scattering experiment automation, incorporating modern AI techniques, enhanced user interfaces, and improved deployment capabilities.

ABSTRACT

ESAC (EQ-SANS Assisting Chatbot) is an advanced AI-powered application designed to streamline the workflow of neutron scattering experiments at the Spallation Neutron Source (SNS). This report documents the significant advancements in ESAC v1, which include the integration of a combined In-Context Learning (ICL) + Retrieval-Augmented Generation (RAG) capability, a robust integrated development environment, and standalone executable distribution. These enhancements address the limitations of the original version, making ESAC v1 a transformative tool for researchers. The report also discusses the technical challenges encountered during development and their resolution, highlighting the impact of these improvements on the neutron scattering research community.

1. INTRODUCTION

Neutron scattering experiments are critical for understanding the structural and dynamic properties of materials. However, the complexity of experiment planning and data analysis often poses challenges for researchers. The original ESAC system, introduced in SoftwareX (Do, Nagy, and Heller 2025), aimed to simplify this process by providing an AI-powered chatbot for generating Python scripts tailored to EQ-SANS experiments. Despite its innovative approach, the initial version had several limitations, including basic AI capabilities, a lack of advanced user interface features, and dependency on Python runtime environments.

ESAC v1 builds upon this foundation, addressing these shortcomings through a series of targeted improvements. By integrating advanced AI modes, enhancing the graphical user interface, and enabling standalone executable distribution, ESAC v1 transforms the application into a comprehensive tool for neutron scattering research. This report provides a detailed account of these advancements, their implementation, and their impact on the research workflow.

2. AI ARCHITECTURE IMPROVEMENTS

2.1 IN-CONTEXT LEARNING (ICL) — PART OF THE COMBINED ICL+RAG APPROACH

A core AI improvement in ESAC v1 is the adoption of In-Context Learning (ICL) as part of a combined ICL+RAG approach. Unlike the original system's basic prompt engineering, the ICL capability assembles a rich, task-specific knowledge payload (up to roughly 150,000 tokens) and injects it into the model's context window when appropriate. This design provides several advantages:

- **Comprehensive knowledge access**: The AI can simultaneously access documentation, function definitions, configuration descriptions, and representative historical scripts.
- **Reduced hallucination**: With authoritative local context available, the AI is less likely to invent APIs, parameters, or workflows.
- **Complex reasoning**: The system can draw connections across experiment planning, data reduction, and safety constraints that span multiple documents.
- **Consistency**: Generated scripts remain aligned with established instrument protocols, naming conventions, and safety procedures.

The ICL knowledge context is the curated set of documents, code, and metadata included in the in-context payload when invoking the LLM in an ICL-style call. It typically includes:

- **Instrument reference material**: manuals, configuration descriptions, and safety procedures for EQ-SANS.
- **API and function documentation**: docstrings and short reference snippets for services such as `ScriptExecutor` and `KnowledgeManager`.
- **Curated scan and script patterns**: canonical scan and reduction fragments (for example, from `eqsans_scanfunctions_live.py`).
- **Historical scripts**: recent, high-quality reduction and scan scripts (e.g., from `/home/controls/var/tmp`).
- **Configuration data**: parsed or summarized configuration files, notably Q-range configurations under `/home/controls/var/QRangeConfigurations/`.
- **Knowledge-base notes**: concise guidance and how-tos from the `knowledge/` directory.
- **Safety and compliance rules**: short, explicit rules that must not be violated by generated scripts.
- **Session context**: a small window of recent chat and editor content capturing the user's immediate intent.

Including this curated context improves grounding and accuracy, ensures consistency with local conventions and safety rules, increases precision for Q-range and configuration calculations by reusing the same parsing logic and constants as the production environment, and reduces RAG retrieval round-trips (lowering latency and cost) by answering many queries directly from the in-context payload. It also enables cross-document reasoning, such as matching a configuration entry to a specific script fragment or beamline workflow.

To keep ICL payloads efficient and safe, ESAC v1 applies several operational safeguards when assembling the context:

- **Curated truncation**: fragments are ranked and trimmed to respect model token limits while preserving critical content.
- **Caching**: frequently used fragments (e.g., standard scan templates and configuration schemas) are cached to speed assembly.
- **Safe filtering**: sensitive information such as API keys, personally identifiable information (PII), and restricted logs is explicitly excluded.
- **RAG fallback**: for unusually large or rare queries, a focused RAG retrieval step is used to append only the most relevant additional documents.

2.2 RETRIEVAL-AUGMENTED GENERATION (RAG) — INTEGRATED WITH ICL

Integrated with the ICL capability, Retrieval-Augmented Generation (RAG) provides targeted retrieval and augmentation for queries that benefit from focused knowledge lookups. The combined approach enables the system to:

- **Smart Query Analysis**: Parses user queries to identify relevant keywords, function names, and technical terms
- **Relevance Scoring**: Scores knowledge base sections based on multiple criteria including exact matches, function references, and technical terminology
- **Context Extraction**: Extracts relevant code sections and documentation with surrounding context
- **Efficient Processing**: Provides faster responses for specific queries while maintaining accuracy

The project's 'knowledge/' directory contains the active, human-readable modules that the RAG system indexes for retrieval: the 'module*.md' files are concise topical references (instrument theory, configuration snippets, analysis recipes, and experimental workflows) written for easy lookup. 'module9.json' provides structured, machine-readable mappings such as named Q-range presets and calibration constants that the retriever can inject into prompts to avoid numeric guessing. The 'rss-eqsans.pdf' file provides a canonical instrument overview for reference. However, these files are hidden if the executable is used.

2.3 MODEL SUPPORT

ESAC v1 uses three production models (available via the openrouter API gateway) selected for a balance of latency, cost, and reasoning capability. The supported models are:

Table 1. Production Models Supported in ESAC v1

oprule Model	Provider	Primary Use Case
gpt-4o-mini	OpenAI	Fast, cost-effective general tasks and interactive assistance
claude-3-haiku	Anthropic	Short, high-quality reasoning with conservative outputs
gemini-2.5-flash	Google	High-throughput generation with strong code/analysis performance

3. USER INTERFACE ENHANCEMENTS

ESAC v1 introduces a modernized and feature-rich user interface that integrates a multi-tab Python script editor with an AI-powered chat panel. A screenshot of the main application window is shown in Figure 1, illustrating the split-pane layout, toolbar controls, model selector, token tracker, and time-estimation tools.

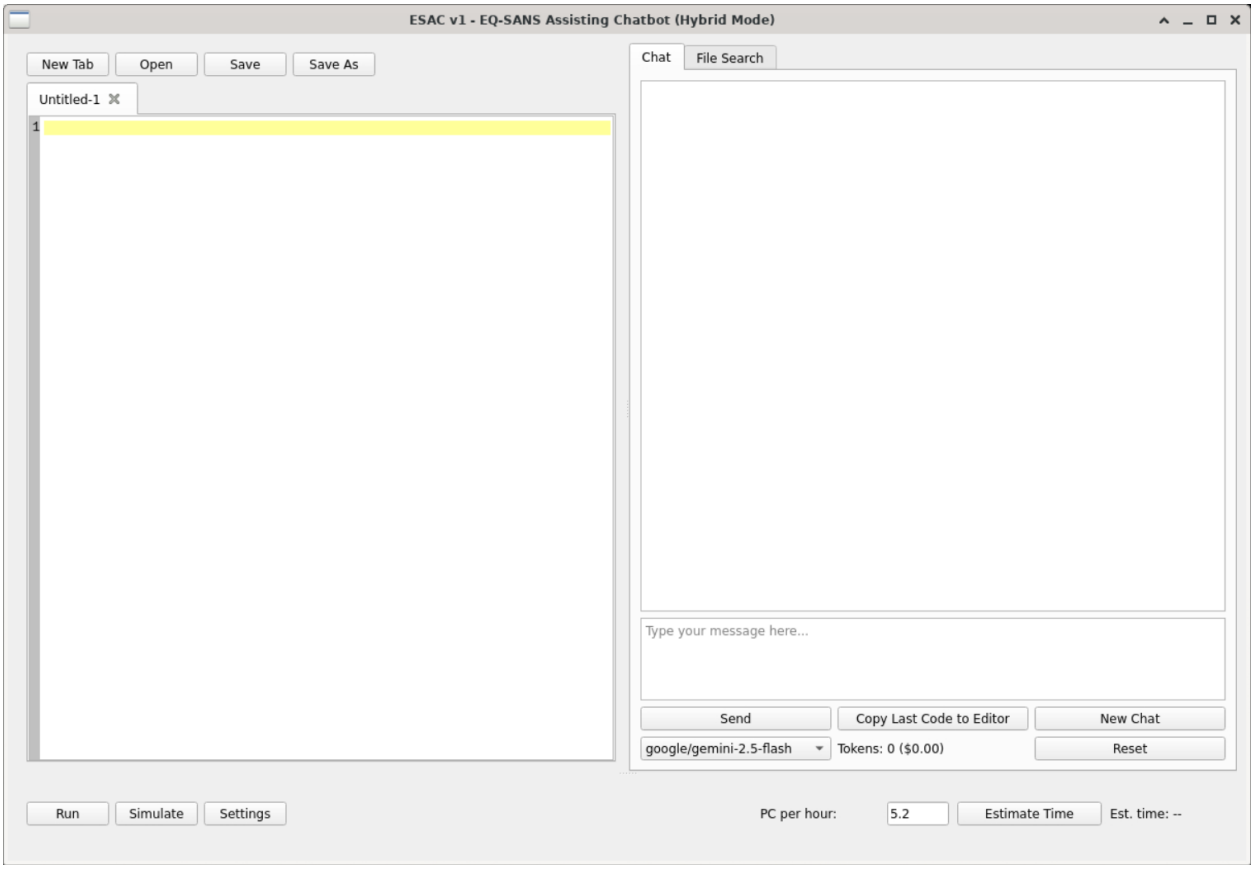

Figure 1. ESAC v1 user interface, showing the multi-tab script editor (left), AI chat assistant (right), and bottom control panel with execution, simulation, model selection, cost tracking, and time estimation tools.

3.1 INTEGRATED SCRIPT EDITOR

ESAC v1 features a comprehensive script editor that transforms the application from a simple chatbot into a complete development environment:

- **Multi-Tab Support**: Users can work with multiple scripts simultaneously.
- **Syntax Highlighting**: Python syntax highlighting for improved readability.
- **Save/Load Functionality**: Persistent storage of scripts with automatic backup.
- **Line Numbers**: Provides a professional code editing experience.
- **Search and Replace**: Supports efficient code editing and refactoring.

3.2 FILE SEARCH AND SCRIPT REUSE

To streamline repeated experimental workflows, ESAC v1 introduces an integrated file-search facility optimized for locating and reloading historical scripts. Key features:

- **Purpose-built for reuse**: Designed to quickly find past scripts (by IPTS, user, filename, or content) so users can reload and re-run them with minimal effort.
- **Non-blocking search**: File-system searches run in background threads to keep the UI responsive and avoid freezes during long scans.
- **Search-to-load flow**: Search results are returned with indexed entries so users can issue simple commands like "load script 3" in the chat to copy the script into the editor.
- **Recency-first ordering**: Results are sorted by modification time so the most recently used or edited scripts appear first.

3.3 ENHANCED CHAT INTERFACE

The chat interface has been significantly improved with several new features:

- **Copy to Editor**: One-click transfer of generated code to the script editor.
- **Conversation History**: Persistent chat history preserved across sessions.
- **Token Tracking**: Real-time monitoring of API usage, token counts, and estimated cost.
- **Model Selection**: Dynamic switching between multiple LLM models.
- **Context Awareness**: Automatically incorporates the active script or user intent when generating responses.

3.4 Q-RANGE CALCULATOR

ESAC v1 includes a specialized Q-Range Calculator integrated into the chat interface to provide instant scattering parameter calculations for EQ-SANS instrument configurations. Highlights:

- **Natural language queries**: Users can ask questions like "what is the Q-range of 4m 2.5a config" and receive computed QMin/QMax, wavelength ranges, TOF windows, and beam diameters.
- **Config-driven**: The calculator loads instrument parameters from configuration files (`.sav` files) stored on the instrument host (default: `/home/controls/var/QRangeConfigurations/`).
- **LLM-assisted matching**: When a direct filename match is ambiguous, the system uses the LLM to match a natural-language query to the best available configuration entry.
- **Frequency handling**: Supports both 30Hz and 60Hz operation modes and selects the appropriate configuration automatically (defaults to 60Hz unless otherwise requested).
- **Error reporting**: If configuration files are missing or unreadable, the UI reports clear, actionable errors and suggests the local path to place configs.

3.5 TIME AND COST ESTIMATION

ESAC v1 provides real-time estimation capabilities:

- **Execution Time**: Estimates experiment runtime based on proton charge rate and script content.
- **API Cost Tracking**: Displays token consumption and cost associated with LLM usage.
- **Model Comparison**: Helps users understand trade-offs between model speed, quality, and cost.
- **Budget Awareness**: Enables users to manage LLM usage responsibly during long experiments.

4. DEPLOYMENT AND DISTRIBUTION

4.1 EXECUTABLE DISTRIBUTION

The most significant deployment improvement is the PyInstaller-based executable distribution:

- **Zero Dependencies**: Users can run ESAC without installing Python or any dependencies
- **SSL Certificate Bundling**: Includes secure HTTPS certificates for API communication
- **Cross-Platform**: Single executable works across Linux, Windows, and macOS
- **Configuration Management**: Secure encrypted storage of API keys and settings
- **Automatic Updates**: Self-contained application with no external dependencies

4.2 KNOWLEDGE BASE MANAGEMENT

ESAC v1 includes sophisticated knowledge base management:

- **Modular Knowledge**: Organized knowledge files for different aspects of EQ-SANS
- **Development Integration**: Automatic loading of development scripts when available
- **Version Control**: Git integration for knowledge base updates
- **Extensibility**: Easy addition of new knowledge domains

4.3 SECURITY AND SENSITIVE CONFIGURATION HANDLING

To reduce the risk of accidental credential leakage, the development and packaging workflow now treats local configuration files and environment variable stores as sensitive:

- **Ignored artifacts**: Local configuration files (for example, a `config/` directory or `env.txt`) are excluded from version control via `.gitignore` entries in the development repository.
- **Local-only keys**: API keys and other secrets should be placed in a local `.env` or `config/env.txt` file and will be encrypted by the application on first run; these files must not be committed to source control.
- **Developer workflow**: If a sensitive file was previously committed, it must be removed from the repository history and the working tree (the release process includes steps to ensure no keys remain in tagged releases).

5. TECHNICAL IMPLEMENTATION

5.1 ARCHITECTURE OVERVIEW

ESAC v1 follows a modular architecture with clear separation of concerns:

- **GUI Layer**: PyQt5-based interface with chat widget and script editor
- **Service Layer**: LLM service, knowledge manager, and configuration manager
- **AI Integration**: OpenRouter API client with streaming support
- **Deployment Layer**: PyInstaller configuration for executable generation

5.2 PERFORMANCE OPTIMIZATIONS

Several performance optimizations have been implemented:

- **Streaming Responses**: Real-time AI response streaming for better user experience
- **Context Caching**: Intelligent caching of knowledge base content
- **Memory Management**: Efficient handling of large knowledge bases
- **Threading**: Non-blocking UI during AI processing

6. DISCUSSION

Table 2. Feature Comparison: ESAC Original vs ESAC v1

Feature	Original ESAC	ESAC v1
AI Mode	Basic prompting	ICL + RAG modes
User Interface	Simple chat	Integrated IDE
Script Editor	None	Multi-tab with syntax highlighting
Model Support	Single model	3 models via OpenRouter
Deployment	Python script	Standalone executable
Cost Tracking	None	Real-time token/cost monitoring
Knowledge Base	Static	Dynamic with relevance scoring
Context Window	Limited	Up to 150K tokens
Save/Load	None	Full project persistence
Time Estimation	None	Proton charge-based calculation

A comparative summary of major feature differences between the original ESAC system and ESAC v1 is provided in Table 2. The new version delivers substantial improvements across AI capability, user interface design, deployment accessibility, and knowledge management. These enhancements are the result of addressing several key technical challenges encountered during development.

One of the most significant challenges was the implementation of the combined In-Context Learning (ICL) and Retrieval-Augmented Generation (RAG) architecture. ICL required careful optimization of the model's context window to accommodate large, curated knowledge payloads without exceeding token limits, while RAG required an efficient retrieval mechanism capable of returning highly relevant information with minimal latency. The final solution uses a hybrid approach supported by intelligent caching, context ranking, and selective augmentation, enabling ESAC v1 to generate more accurate, grounded, and consistent outputs.

User experience also improved dramatically over the original version. The introduction of a multi-tab script editor, syntax highlighting, integrated search, and real-time cost tracking transformed ESAC from a simple chatbot into a full-featured development environment. These features allow researchers to manage multiple reduction or scan scripts simultaneously, perform side-by-side comparisons, and maintain persistent project context across sessions.

Deployment presented another major improvement area. The adoption of PyInstaller for packaging ESAC v1 as a standalone executable eliminated the need for Python environments or dependency management, significantly simplifying installation for instrument scientists and visiting users. Secure storage of API keys and automated update handling further strengthened the reliability and portability of the system.

Collectively, these enhancements have made ESAC v1 a far more capable and accessible tool for neutron scattering research. Early adopters report substantial reductions in script preparation time, fewer errors due to improved grounding of generated code, and a more intuitive workflow overall. The integration of advanced AI reasoning with a modernized interface and robust deployment strategy positions ESAC v1 as a foundational tool for future automation across SNS instruments.

7. REFERENCES

Do, Changwoo, Gergely Nagy, and William T. Heller. 2025. "ESAC (EQ-SANS Assisting Chatbot): Application of large language models and retrieval-augmented generation for enhanced user experience at EQ-SANS." *SoftwareX* 31:102191. ISSN: 2352-7110. https://doi.org/https://doi.org/10.1016/j.softx.2025.102191. https://www.sciencedirect.com/science/article/pii/S235271102500158X.

APPENDIX A. INSTALLATION INSTRUCTIONS & CONFIGURATION

APPENDIX A. INSTALLATION INSTRUCTIONS & CONFIGURATION

A.1 DEMO VIDEO

Refer to ESAC V.1 Demo for video showing demonstration.

A.2 FROM SOURCE

The ESAC v1 application can be run directly from source on Linux, macOS, or Windows systems with Python 3.8 or later.

1. Clone the repository:

```
git clone https://github.com/cw-do/esac-v1.git
cd esac-v1
```

2. (Optional but recommended) Create and activate a virtual environment:

```
python -m venv esac_env
source esac_env/bin/activate    # Windows: esac_env\Scripts\activate
```

3. Install dependencies:

```
pip install -r requirements.txt
```

4. Configure the API key:

   Edit `config/env.txt` and set:

   OPENROUTER_API_KEY=your_api_key_here

   The key will be encrypted and stored securely on first launch.

5. Run the application:

```
python main.py
```

6. (Optional) Launch with a large font size:

```
python main.py --largefont
```

7. (Optional) Show tokens used:

```
python main.py --showtoken
```

A.3 USING EXECUTABLE

Pre-built standalone executables may be provided for Linux, Windows, or macOS. These require no Python installation or external dependencies.

1. Download the appropriate executable from the GitHub Releases page.
2. Make the file executable (Linux/macOS):

```
chmod +x main
```

3. Run the executable directly:

```
./main
```

4. On first run, the settings dialog will prompt for an API key.

A.4 API KEY SETUP

1. Obtain an API key from https://openrouter.ai/
2. Launch ESAC v1
3. The application will prompt for API key configuration
4. Keys are encrypted and stored securely

A.5 MODEL CONFIGURATION

Users can select from multiple AI models through the interface dropdown. Each model has different performance characteristics:

- **GPT-4o-mini**: Fastest, most cost-effective
- **GPT-4o**: Best for complex code generation
- **Claude-3-Sonnet**: Excellent for reasoning tasks