Article

MR-GWAS-JSON: A web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation

WenJun Zhang
School of Life Sciences, Sun Yat-sen University, Guangzhou, China
E-mail: zhwj@mail.sysu.edu.cn, wjzhang@iaees.org

Received 16 December 2025; *Accepted 11 January 2026*; *Published online 25 February 2026*; *Published 1 December 2026*

Abstract
Mendelian Randomization analysis requires high-quality genetic variant data (SNPs) for exposure and outcome traits, aligned alleles, and a correlation matrix for statistical modeling. The MR-GWAS-JSON data fetcher, developed in present study, is a web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation. It assists researchers in preparing harmonized MR datasets. It integrates direct JSON validation, file upload processing, AI-powered GWAS generation, multi-source database fetching, advanced caching, retry logic, and rate limiting. Overall, MR-GWAS-JSON can be used as an easy tool to obtain the data for MR analysis.

Keywords GWAS (Genome-Wide Association Study); JSON; data fetching; Mendelian Randomization (MR); Artificial Intelligence (AI); Large Language Models (LLMs); web tool.

Network Biology
ISSN 2220-8879
URL: http://www.iaees.org/publications/journals/nb/online-version.asp
RSS: http://www.iaees.org/publications/journals/nb/rss.xml
E-mail: networkbiology@iaees.org
Editor-in-Chief: WenJun Zhang
Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction

Mendelian randomization (MR) represents a powerful epidemiological approach to infer causality in observational data by leveraging genetic variants as instrumental variables, mimicking the randomization in controlled trials (Burgess et al., 2013; Burgess and Bowden, 2015; Burgess et al., 2016; Hartwig et al., 2017; Zhang, 2025b). At its core, MR exploits the random assortment of alleles during meiosis to estimate causal effects of exposures on outcomes, particularly when randomized trials are infeasible or unethical. The method assumes three key principles: the genetic variant (instrument) strongly associates with the exposure, remains independent of confounders, and affects the outcome solely through the exposure (GWASLab, 2024). Single nucleotide polymorphisms (SNPs) from genome-wide association studies (GWAS) serve as primary instruments, providing robust evidence against confounding and reverse causation. Early methodological advancements focused on integrating summarized GWAS data from multiple variants to enhance precision,

such as inverse-variance weighted estimators that address bias from weak instruments or pleiotropy (Burgess et al., 2013; Burgess and Bowden, 2015). Robust extensions, like those using multiple candidate instruments, further mitigate violations of instrument validity, ensuring reliable causal inference in fields like cardiovascular disease and oncology (Burgess et al., 2016; Hartwig et al., 2017). The success of MR hinges on high-quality genetic data, underscoring the need for accurate SNP-phenotype associations to power these analyses (Zhang, 2025a-b).

Genome-wide association studies (GWAS) form the foundational data source for MR, systematically scanning genomes to identify SNPs associated with traits, diseases, or biomarkers (Visscher et al., 2012; Tam et al., 2019; Zhang and Qi, 2026). GWAS data encapsulate the effects of SNPs on phenotypes, offering insights into genotype-phenotype associations that reveal genetic architectures underlying complex traits (Zhang, 2025a-b, 2026a-b). These studies have revolutionized genetics by pinpointing thousands of loci, yet challenges persist, including polygenic signals, population stratification, and the need for large sample sizes to detect rare variants (Tam et al., 2019). In MR contexts, GWAS summary statistics—reporting SNP-exposure and SNP-outcome associations—are aggregated to construct instrumental variables, enabling efficient analyses without individual-level data (Burgess et al., 2013; Davies et al., 2018). Recent tools have streamlined GWAS data access; for instance, an earlier web-based fetcher integrates multiple public databases to retrieve genotype, phenotype, and summary statistics, providing a reliable pipeline despite occasional access restrictions (Zhang, 2026b). This approach ensures comprehensive data for MR, where SNPs significantly linked to exposures (e.g., lipid levels or inflammation markers) are selected to probe causal pathways (GWASLab, 2024).

The integration of artificial intelligence (AI) into GWAS and MR workflows is transforming data handling, analysis, and interpretation, addressing limitations in scale and complexity (Rajkomar et al., 2019; Topol, 2019; Zhang, 2025a, 2026a, c-g; Zhang and Qi, 2026). AI excels at processing vast, heterogeneous genomic datasets, uncovering patterns that traditional methods overlook, such as subtle pleiotropic effects in MR (Hemani et al., 2018). For GWAS, AI-driven tools automate data fetching and generation, bypassing ethical hurdles in synthetic medical data creation (Extance, 2025). A notable example is a web-based GWAS data fetcher powered by AI models like DeepSeek, Google Gemini, and OpenAI GPT, which synthesizes genetic variant data for MR analysis from diverse sources (Zhang, 2026a). This tool not only retrieves but also generates high-fidelity summary statistics, enhancing accessibility for researchers facing database constraints (Zhang and Qi, 2026). Similarly, AI facilitates web-based data generators tailored for MR, simulating instrumental variables and exposure-outcome associations to validate methods or impute missing data (Zhang, 2025a). Beyond retrieval, AI employs natural language processing (NLP) for literature synthesis, using named entity recognition (NER) to extract biological entities like genes and species from publications, then building knowledge graphs for integrated insights (Zhang, 2026g).

The rapid advancement of artificial intelligence (AI) is poised to transform how we tackle complex global challenges. Modern AI systems can integrate, analyze, and derive insights from massive, heterogeneous datasets—a task that has traditionally surpassed the limits of conventional analysis (Zhang, 2025a, 2026a, c-g; Zhang and Qi, 2026). Leveraging sophisticated machine learning and real-time data processing, AI uncovers hidden patterns and produces actionable, high-fidelity intelligence. Its proven success across fields from precision medicine to supply chain optimization demonstrates this transformative potential. In practice, AI serves as a powerful tool for systematically searching, comprehending, and synthesizing the vast and expanding corpus of ecological and taxonomic literature. Using advanced natural language processing (NLP) and knowledge-retrieval engines, it accesses and interprets global repositories, scientific publications, and curated databases. Core to this process are key technical steps (Zhang, 2025a, 2026a, c-g; Zhang and Qi, 2026), including Named Entity Recognition (NER) to identify biological entities (e.g., species, genes) and statistical

data extraction to parse results from unstructured text and tables. This information is then structured into an actionable knowledge graph. Crucially, AI moves beyond basic retrieval to perform advanced synthesis: dynamically connecting findings across multiple sources, cross-referencing studies, and integrating disparate evidence to form a continuously evolving, unified knowledge base.

AI's role extends to specialized MR applications, including AI-based platforms for medication consultation and trait analysis, which incorporate GWAS-derived insights for personalized causal predictions (Zhang, 2026d-e, g). For instance, tools like TraitGenePathAna use AI to dissect biological traits via genetic pathways, aiding MR in environmental adaptation studies (Zhang, 2026b-c, f). In climate change research, AI assesses species adaptation using physiological and genetic indicators from GWAS, informing causal MR on resilience factors (Zhang, 2026b-c). Ethically, AI-generated data for MR sidesteps traditional reviews, accelerating innovation while raising oversight needs (Extance, 2025). Overall, AI augments MR and GWAS by enhancing data quality, reducing biases, and enabling scalable causal inference, with future potential in real-time genomic surveillance. These synergies promise to deepen our understanding of gene-environment interactions, driving advancements in public health and beyond.

Integrated and synthesized from the previous works (Zhang, 2025a, 2026a; Zhang and Qi, 2026), this study aims to present a web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation, which can be used as an easy way to obtain the necessary data for MR analysis.

2 Overview

MR-GWAS-JSON data fetcher is a web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation.

2.1 Purpose

The tool assists researchers in preparing **harmonized MR datasets**. Mendelian Randomization requires high-quality genetic variant data (SNPs) for **exposure** and **outcome** traits, aligned alleles, and a correlation matrix for statistical modeling.

It integrates:

- Direct JSON validation
- File upload processing
- AI-powered GWAS generation
- Multi-source database fetching
- Advanced caching, retry logic, and rate limiting

2.2 Key Value

Streamlined MR dataset preparation in a **browser-based interface**

AI-assisted correlation matrix estimation

Multi-database support for GWAS summary statistics

Quality control and advanced validation for SNP data

2.3 Functionality

The UI provides **three workflows**:

Workflow 1 — Direct JSON Input

- Paste harmonized JSON directly.
- Structured validation for MR fields (snps, betaX, betaY, etc.).
- Displays formatted JSON in an output panel.

Workflow 2 — File Upload & Harmonization

- Upload exposure/outcome data in CSV, TXT, JSON, .gz.

- Parse & normalize field names.
- Harmonize datasets (align effect/non-effect alleles; flip sign if reversed).
- Fetch correlation matrix from:
  - **AI services** (DeepSeek, Gemini)
  - **Public databases** (LDlink, 1000 Genomes, gnomAD)
- Output MR-ready JSON.

Workflow 3 — AI & Multi-Source Solutions

Option A — AI-Powered GWAS

- Enter trait names (exposure & outcome).
- AI generates variant-level genetic associations.
- Harmonize datasets & fetch correlation matrix.

Option B — Multi-Source GWAS Fetcher

- Query multiple genetic databases (IEU OpenGWAS, GWAS Catalog, FinnGen, GWAS Atlas, eQTLGen).
- Filter SNPs by p-value threshold.
- Harmonize fetched datasets & generate correlation matrix.

3 Algorithmic Description

3.1 Core Steps

The process flow can be visualized like this:

User Input → Data Fetch/Upload → Parse & Normalize → Harmonize → Correlation Matrix → JSON Output

Step 1 — Parsing Files

Uses parseFile() to handle multiple formats & compression:

```
js
async function parseFile(file) {
    const buf = await file.arrayBuffer();
    let text;

    // Handle gzip
    if (file.name.endsWith('.gz') || isGzip(buf)) {
        text = pako.ungzip(new Uint8Array(buf), { to: 'string' });
    } else {
        text = new TextDecoder().decode(buf);
    }

    // Try JSON first
    if (file.name.endsWith('.json') || text.trim().startsWith('{')) {
        return normalizeJsonData(JSON.parse(text));
    }

    // Otherwise parse CSV/TSV
    return parseCsvData(text);
}
```

- Detects .gz

- Attempts JSON parsing first
- Falls back to CSV parsing

Step 2 — Normalizing Rows

Maps various column names to standard MR fields:

```
js
function normalizeRow(row) {
    const fieldMap = {
        rsid: ['snp', 'rsid', 'variant', 'markername', 'id'],
        ea:   ['effect_allele', 'ea', 'a1'],
        nea:  ['other_allele', 'nea', 'a2'],
        beta: ['beta', 'effect_size'],
        se:   ['se', 'stderr'],
        eaf:  ['eaf', 'maf']
    };

    const lowerRow = {};
    for (let key in row) lowerRow[key.toLowerCase()] = row[key];

    const result = {};
    for (let target in fieldMap) {
        for (let candidate of fieldMap[target]) {
            if (candidate in lowerRow && lowerRow[candidate]) {
                result[target] = lowerRow[candidate];
                break;
            }
        }
    }

    // Convert numbers
    if (result.beta) result.beta = Number(result.beta);
    if (result.se) result.se = Number(result.se);
    if (result.eaf) result.eaf = Number(result.eaf);

    return result;
}
```

- Handles heterogeneous GWAS file field labels
- Converts numeric fields

Step 3 — Harmonization

Aligns alleles & flips outcome beta if effect allele is reversed:

```
js
function harmonizeData(exposureData, outcomeData) {
    const commonRsids = Object.keys(exposureData).filter(rsid => outcomeData[rsid]);
    const snps = [], betaX = [], betaY = [], betaXse = [], betaYse = [];
```

```
    for (const rsid of commonRsids) {
        let betaYValue = outcomeData[rsid].beta;
        if (exposureData[rsid].ea === outcomeData[rsid].nea &&
            exposureData[rsid].nea === outcomeData[rsid].ea) {
            betaYValue = -betaYValue;
        }
        snps.push(rsid);
        betaX.push(exposureData[rsid].beta);
        betaY.push(betaYValue);
        betaXse.push(exposureData[rsid].se);
        betaYse.push(outcomeData[rsid].se);
    }
    return { snps, betaX, betaY, betaXse, betaYse };
}
```

Step 4 — Correlation Matrix

Two fetching strategies:

- **fetchAICorrelationMatrix()** — prompts AI with SNP list to return realistic LD-based matrix
- **fetchCorrelationMatrix()** — queries LDlink/1000Genomes/gnomAD APIs

Fallback: Identity matrix if correlation fetch fails.

Step 5 — JSON Output Generation

Final JSON includes:

```
json
{
  "snps": [...],
  "betaX": [...],
  "betaY": [...],
  "betaXse": [...],
  "betaYse": [...],
  "correlation": [[1,...],[...]],
  "exposure": "BMI",
  "outcome": "Type 2 Diabetes",
  "metadata": {
    "harmonized_snps": 15,
    "correlation_source": "deepseek",
    "timestamp": "2026-02-18T15:03:12Z"
  }
}
```

3.2 Modules/Procedures

Workflow 1 — Direct JSON Input

1. Paste harmonized JSON in textarea.
2. Click “Validate JSON”.
3. Tool checks presence & array length of MR fields.
4. Displays pretty-printed JSON in output area.

Workflow 2 — File Upload & Harmonization

1. Upload exposure & outcome files.
2. Tool parses and normalizes variant fields.
3. Click “Harmonize Data”.
4. Choose correlation source (AI or public).
5. Tool fetches correlation matrix.
6. Final JSON is generated for download/copy.

Workflow 3 — AI & Multi-Source

Option A — AI-Powered GWAS

1. Enter traits & API key.
2. Tool calls AI service with structured prompt.
3. AI returns CSV-formatted exposure & outcome variants.
4. Harmonization + correlation fetching.
5. JSON output generated.

Option B — Multi-Source GWAS

1. Select databases & optional API tokens.
2. Enter traits & p-value threshold.
3. Tool queries APIs sequentially with **RateLimiter** & **DataCache**.
4. Harmonization + correlation fetching.
5. JSON output generated.

3.3 Sample JS Snippets

Rate Limiter

```
js
class RateLimiter {
    constructor(maxRequests, perSeconds) {
        this.maxRequests = maxRequests;
        this.perSeconds = perSeconds;
        this.requests = [];
    }
    async waitForSlot() {
        const now = Date.now();
        const cutoff = now - this.perSeconds * 1000;
        this.requests = this.requests.filter(t => t > cutoff);
        if (this.requests.length >= this.maxRequests) {
            const waitTime = this.requests[0] + this.perSeconds * 1000 - now;
            await new Promise(r => setTimeout(r, waitTime));
        }
        this.requests.push(now);
    }
}
```

Retry Logic

```
js
async function fetchWithRetry(url, options, maxRetries = 3) {
    let lastErr;
```

```
    for (let i = 0; i < maxRetries; i++) {
        try {
            return await fetch(url, options);
        } catch (err) {
            lastErr = err;
            await new Promise(r => setTimeout(r, 1000 * (i+1)));
        }
    }
    throw lastErr;
}
```

4 User Guide

4.1 Launching

1. Open advancedJSON.txt in browser.
2. Choose workflow.

Workflow 1

- Paste JSON → Validate → Copy/Download.

Workflow 2

- Upload files → Harmonize → Fetch correlation → Download JSON.

Workflow 3

- Choose AI/Multi-source option → Configure → Fetch → Harmonize → Output JSON.

4.2 Tips

- Ensure allele fields are correct for harmonization.
- AI keys must match service selected.
- For large SNP sets, correlation fetching can be slow — identity matrix fallback available.

5 Complete Codes

The following are the HTML+JavaScript codes of the tool (http://www.iaees.org/publications/journals/nb/articles/2026-16(4)/MR-GWAS-JSON.htm; Fig. 1):

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="author" content="W. J. Zhang" />
<link href="../../../../../style.css" rel="stylesheet" media="screen" type="text/css">
<title> MR-GWAS-JSON: A web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Fetching MR Data</title>
    <style>
        * {
            margin: 0;
```

```
        padding: 0;
        box-sizing: border-box;
    }
    body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        background: white; /* Changed from white to white */
        min-height: 100vh;
        padding: 20px;
    }
    .container {
        max-width: 1200px;
        margin: 0 auto;
        background: white; /* Changed from white to white */
        border-radius: 15px;
        padding: 30px;
        box-shadow: 0 10px 40px rgba(0,0,0,0.3);
    }
    h1 {
        color: #667eea;
        margin-bottom: 10px;
        font-size: 28px;
    }
    .subtitle {
        color: #666;
        margin-bottom: 20px;
        font-size: 14px;
    }
    .workflow-choice {
        display: grid;
        grid-template-columns: repeat(auto-fit, minmax(280px, 1fr));
        gap: 20px;
        margin: 30px 0;
    }
    .choice-card {
        border: 3px solid #e0e0e0;
        border-radius: 12px;
        padding: 20px;
        cursor: pointer;
        transition: all 0.3s;
        background: white; /* Changed from #fafafa to white */
    }
    .choice-card:hover {
        border-color: #667eea;
        background: #f0f7ff;
        transform: translateY(-5px);
```

```
        box-shadow: 0 5px 20px rgba(102, 126, 234, 0.2);
    }
    .choice-card.active {
        border-color: #667eea;
        background: #f0f7ff;
        box-shadow: 0 5px 20px rgba(102, 126, 234, 0.3);
    }
    .choice-icon {
        font-size: 3rem;
        margin-bottom: 15px;
        display: block;
        text-align: center;
    }
    .choice-card h3 {
        color: #667eea;
        margin-bottom: 10px;
        font-size: 1.2rem;
    }
    .choice-card p {
        color: #666;
        line-height: 1.5;
        font-size: 14px;
    }
    .workflow-section {
        display: none;
        margin-top: 30px;
        padding: 20px;
        border: 2px solid #e0e0e0;
        border-radius: 12px;
        background: white; /* Changed from #f9f9f9 to white */
    }
    .workflow-section.active {
        display: block;
    }
    .form-group {
        margin-bottom: 20px;
    }
    label {
        display: block;
        margin-bottom: 8px;
        color: #333;
        font-weight: 600;
        font-size: 14px;
    }
    input[type="text"],
```

```
input[type="password"],
input[type="number"],
select,
textarea {
    width: 100%;
    padding: 12px;
    border: 2px solid #e0e0e0;
    border-radius: 8px;
    font-size: 14px;
    transition: border-color 0.3s;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: white; /* Changed from white to white */
}
textarea {
    font-family: 'Courier New', monospace;
    font-size: 13px;
    resize: vertical;
    min-height: 300px;
}
textarea.output-json {
    background: white; /* Changed from #f0f0f0 to white */
    min-height: 400px;
}
input:focus, select:focus, textarea:focus {
    outline: none;
    border-color: #667eea;
}
.btn {
    padding: 12px 24px;
    border: none;
    border-radius: 8px;
    font-size: 14px;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s;
    margin-right: 10px;
    margin-bottom: 10px;
}
.btn-primary {
    background: #667eea;
    color: white;
}
.btn-primary:hover:not(:disabled) {
    background: #5568d3;
    transform: translateY(-2px);
```

```
}
.btn-secondary {
    background: #f0f0f0;
    color: #333;
}
.btn-secondary:hover {
    background: #e0e0e0;
}
.btn-success {
    background: #28a745;
    color: white;
}
.btn-success:hover:not(:disabled) {
    background: #218838;
}
.btn-danger {
    background: #dc3545;
    color: white;
}
.btn-danger:hover {
    background: #c82333;
}
.btn:disabled {
    background: #ccc;
    cursor: not-allowed;
    transform: none;
}
.file-input-group {
    border: 2px dashed #ccc;
    border-radius: 8px;
    padding: 20px;
    text-align: center;
    background: white; /* Changed from #f9f9f9 to white */
    margin-top: 10px;
}
.file-input-group:hover {
    border-color: #667eea;
    background: #f0f7ff;
}
.status {
    padding: 10px;
    border-radius: 6px;
    margin-top: 10px;
    font-size: 14px;
}
```

```
.status.success {
    background: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
}
.status.error {
    background: #f8d7da;
    color: #721c24;
    border: 1px solid #f5c6cb;
}
.status.warning {
    background: #fff3cd;
    color: #856404;
    border: 1px solid #ffeaa7;
}
.status.info {
    background: #d1ecf1;
    color: #0c5460;
    border: 1px solid #bee5eb;
}
.loading {
    display: none;
    text-align: center;
    margin: 20px 0;
}
.spinner {
    border: 4px solid #f3f3f3;
    border-top: 4px solid #667eea;
    border-radius: 50%;
    width: 50px;
    height: 50px;
    animation: spin 1s linear infinite;
    margin: 0 auto 10px;
}
@keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
}
.exposure-container {
    border: 2px dashed #e0e0e0;
    border-radius: 8px;
    padding: 15px;
    margin-top: 10px;
    background: white; /* Changed from #fafafa to white */
}
```

```
.exposure-input {
    display: flex;
    gap: 10px;
    margin-bottom: 10px;
    align-items: center;
}
.exposure-input input {
    flex: 1;
}
.radio-group {
    display: flex;
    gap: 20px;
    margin-top: 8px;
}
.radio-label {
    display: flex;
    align-items: center;
    gap: 8px;
    cursor: pointer;
}
input[type="radio"] {
    width: 18px;
    height: 18px;
    cursor: pointer;
}
.two-column {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 20px;
}
.example-data {
    font-size: 11px;
    color: #666;
    margin-top: 8px;
    padding: 8px;
    background: white; /* Changed from #f8f9fa to white */
    border-radius: 4px;
    border-left: 3px solid #667eea;
}
.api-info {
    font-size: 12px;
    color: #666;
    margin-top: 5px;
    padding: 8px;
    background: white; /* Changed from #f0f7ff to white */
```

```
        border-radius: 4px;
    }
    .harmonization-progress {
        background: white; /* Changed from #f8f9fa to white */
        border: 1px solid #dee2e6;
        border-radius: 8px;
        padding: 15px;
        margin: 20px 0;
    }
    .progress-step {
        display: flex;
        align-items: center;
        margin-bottom: 10px;
        padding: 10px;
        border-radius: 6px;
    }
    .progress-step.pending {
        background: white; /* Changed from #f8f9fa to white */
        color: #6c757d;
    }
    .progress-step.active {
        background: #fff3cd;
        color: #856404;
    }
    .progress-step.complete {
        background: #d4edda;
        color: #155724;
    }
    .progress-step.error {
        background: #f8d7da;
        color: #721c24;
    }
    .step-icon {
        margin-right: 10px;
        font-size: 1.2rem;
    }
    .back-btn {
        margin-top: 20px;
    }
    .correlation-info {
        background: white; /* Changed from #e7f3ff to white */
        border: 1px solid #b3d9ff;
        border-radius: 8px;
        padding: 12px;
        margin: 15px 0;
```

```
        font-size: 13px;
    }
    .correlation-info strong {
        color: #004085;
    }
    .correlation-tabs {
        display: flex;
        gap: 10px;
        margin: 15px 0;
        border-bottom: 2px solid #e0e0e0;
    }
    .correlation-tab {
        padding: 10px 15px;
        background: #f0f0f0;
        border: none;
        cursor: pointer;
        border-radius: 6px 6px 0 0;
        font-weight: 600;
        transition: all 0.3s;
    }
    .correlation-tab.active {
        background: #667eea;
        color: white;
    }
    .correlation-tab:hover {
        background: #e0e0e0;
    }
    .correlation-tab.active:hover {
        background: #5568d3;
    }
    .correlation-content {
        display: none;
        padding: 15px;
        background: white; /* Changed from #f9f9f9 to white */
        border-radius: 0 8px 8px 8px;
        margin-bottom: 15px;
    }
    .correlation-content.active {
        display: block;
    }
    .ai-correlation-tabs {
        display: flex;
        gap: 10px;
        margin: 15px 0;
        border-bottom: 2px solid #e0e0e0;
```

```
}
.ai-correlation-tab {
    padding: 10px 15px;
    background: #f0f0f0;
    border: none;
    cursor: pointer;
    border-radius: 6px 6px 0 0;
    font-weight: 600;
    transition: all 0.3s;
}
.ai-correlation-tab.active {
    background: #667eea;
    color: white;
}
.ai-correlation-tab:hover {
    background: #e0e0e0;
}
.ai-correlation-tab.active:hover {
    background: #5568d3;
}
.ai-correlation-content {
    display: none;
    padding: 15px;
    background: white; /* Changed from #f9f9f9 to white */
    border-radius: 0 8px 8px 8px;
    margin-bottom: 15px;
}
.ai-correlation-content.active {
    display: block;
}
/* Option 4 specific styles */
.checkbox-group {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
    gap: 8px;
    margin-top: 8px;
}
.checkbox-item {
    display: flex;
    align-items: center;
    gap: 6px;
}
.checkbox-item input[type="checkbox"] {
    width: auto;
    cursor: pointer;
```

```
}
.checkbox-item label {
    margin: 0;
    font-weight: normal;
    cursor: pointer;
}
.info-box {
    background: white; /* Changed from #e8f4f8 to white */
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 12px;
    border-left: 3px solid #3498db;
    font-size: 11px;
}
.help-text {
    font-size: 10px;
    color: #7f8c8d;
    margin-top: 4px;
    font-style: italic;
}
.api-token-group {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 12px;
    margin-top: 10px;
}
.api-token-item {
    display: flex;
    flex-direction: column;
}
.api-token-item label {
    font-size: 10px;
    margin-bottom: 3px;
}
.api-token-item input {
    font-size: 10px;
    padding: 6px;
}
.result-item {
    background: white; /* Changed from white to white */
    padding: 12px;
    border-radius: 6px;
    margin-bottom: 10px;
    border: 1px solid #e0e0e0;
    box-shadow: 0 2px 4px rgba(0,0,0,0.05);
```

```
}
.result-item h3 {
    color: #667eea;
    margin-bottom: 8px;
    font-size: 14px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}
.result-item p {
    margin: 4px 0;
    font-size: 11px;
}
.data-box {
    background: white; /* Changed from #f8f9fa to white */
    border: 1px solid #ddd;
    border-radius: 4px;
    padding: 10px;
    margin-top: 10px;
    max-height: 300px;
    overflow-y: auto;
    font-family: 'Courier New', monospace;
    font-size: 10px;
    white-space: pre;
}
.download-button {
    background: linear-gradient(135deg, #3498db 0%, #2980b9 100%);
    padding: 5px 12px;
    font-size: 10px;
    margin-top: 5px;
    display: inline-block;
    color: white;
    text-decoration: none;
    border-radius: 4px;
}
.download-button:hover {
    box-shadow: 0 5px 15px rgba(52, 152, 219, 0.4);
}
#progressBar {
    width: 100%;
    height: 25px;
    background-color: white; /* Changed from #ecf0f1 to white */
    border-radius: 12px;
    overflow: hidden;
    box-shadow: inset 0 2px 4px rgba(0,0,0,0.1);
```

```
}
#progressFill {
    height: 100%;
    background: linear-gradient(90deg, #667eea 0%, #764ba2 100%);
    width: 0%;
    transition: width 0.3s ease;
    display: flex;
    align-items: center;
    justify-content: center;
    color: white;
    font-weight: bold;
    font-size: 11px;
}
/* New styles for AI and Multi-Source Solutions */
.solution-tabs {
    display: flex;
    gap: 10px;
    margin: 15px 0;
    border-bottom: 2px solid #e0e0e0;
}
.solution-tab {
    padding: 10px 15px;
    background: #f0f0f0;
    border: none;
    cursor: pointer;
    border-radius: 6px 6px 0 0;
    font-weight: 600;
    transition: all 0.3s;
}
.solution-tab.active {
    background: #667eea;
    color: white;
}
.solution-tab:hover {
    background: #e0e0e0;
}
.solution-tab.active:hover {
    background: #5568d3;
}
.solution-content {
    display: none;
    padding: 15px;
    background: white; /* Changed from #f9f9f9 to white */
    border-radius: 0 8px 8px 8px;
    margin-bottom: 15px;
```

```
        }
        .solution-content.active {
            display: block;
        }
        @media (max-width: 768px) {
            .workflow-choice {
                grid-template-columns: 1fr;
            }
            .two-column {
                grid-template-columns: 1fr;
            }
            .correlation-tabs {
                flex-direction: column;
            }
            .ai-correlation-tabs {
                flex-direction: column;
            }
            .checkbox-group {
                grid-template-columns: 1fr;
            }
            .api-token-group {
                grid-template-columns: 1fr;
            }
            .solution-tabs {
                flex-direction: column;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Fetching MR Data</h1>
        <p class="subtitle">Generate harmonized JSON data for Mendelian Randomization analysis</p>
        <a href="">Zhang WJ. 2026. MR-GWAS-JSON: A web tool for multi-source Mendelian Randomization (MR) data fetching, harmonization, and JSON generation. Network Biology, 16(4): 403-488</a><br><br>
        <!-- Workflow Choice Selection -->
        <div id="workflowChoice">
            <h2 style="color: #333; margin-bottom: 20px;">Choose Your Workflow:</h2>
            <div class="workflow-choice">
                <!-- Choice 1: Direct JSON Input -->
                <div class="choice-card" data-workflow="json">
                    <span></span>
                    <h3>Option 1: Direct JSON Input</h3>
                    <p>Already have harmonized JSON data? Paste it directly and you're done. Perfect for pre-processed datasets.</p>
```

```
                    </div>
                    <!-- Choice 2: File Upload & Harmonization -->
                    <div class="choice-card" data-workflow="files">
                        <span></span>
                        <h3>Option 2: File Upload & Harmonization</h3>
                        <p>Upload exposure/outcome files (CSV, TXT, JSON, GZ), harmonize with correlation matrix from AI or public databases, get JSON output.</p>
                    </div>
                    <!-- Choice 3: AI and Multi-Source Solutions -->
                    <div class="choice-card" data-workflow="ai">
                        <span></span>
                        <h3>Option 3: AI and Multi-Source Solutions</h3>
                        <p>Use AI to fetch GWAS data by trait names OR fetch real GWAS data from multiple databases (IEU OpenGWAS, GWAS Catalog, FinnGen, GWAS Atlas, eQTLGen).</p>
                    </div>
                </div>
            </div>
            <!-- Workflow 1: Direct JSON Input -->
            <div id="workflow-json" class="workflow-section">
                <h2>Direct JSON Input</h2>
                <p class="subtitle">Paste your existing harmonized JSON data below:</p>
                <div class="form-group">
                    <label for="jsonDirectInput">JSON Data</label>
                    <textarea id="jsonDirectInput" placeholder='Paste your JSON here. An example:
{
  "snps": ["rs9939609", "rs1801133", "rs7903146"],
  "betaX": [0.045, 0.032, 0.051],
  "betaY": [0.156, 0.089, 0.234],
  "betaXse": [0.008, 0.007, 0.009],
  "betaYse": [0.045, 0.038, 0.052],
  "correlation": [[1, 0, 0], [0, 1, 0], [0, 0, 1]],
  "exposure": "BMI",
  "outcome": "Type2Diabetes"
}'></textarea>
                    <div class="example-data">
                        <strong>Expected Format:</strong> JSON object with keys: snps (array), betaX, betaY, betaXse, betaYse (arrays), correlation (2D array), exposure, outcome (strings). Alternatively, you can directly edit the JSON data in the box.
                    </div>
                </div>
                <button class="btn btn-primary" id="validateJsonBtn">Validate JSON</button>
                <button class="btn btn-secondary back-btn" onclick="backToChoice()">Back to Choices</button>
                <div id="jsonStatus" class="status" style="display: none;"></div>
            </div>
            <!-- Workflow 2: File Upload & Harmonization -->
```

```
<div id="workflow-files" class="workflow-section">
    <h2>File Upload & Harmonization</h2>
    <p class="subtitle">Upload your exposure and outcome files, then harmonize with correlation matrix:</p>
    <!-- Step 1: File Upload -->
    <div class="form-group">
        <h3 style="color: #667eea; margin-bottom: 15px;">Step 1: Upload Files</h3>
        <div class="two-column">
            <div>
                <label for="exposureFile">Exposure File</label>
                <div class="file-input-group">
                    <p>Click or drag to upload</p>
                    <input type="file" id="exposureFile" accept=".csv,.txt,.json,.gz" />
                </div>
                <div class="example-data">
                    <strong>CSV format:</strong> SNP,ea,nea,beta,se,eaf<br>
                    <strong>JSON format:</strong> {"rs9939609": {"ea":"A","nea":"T","beta":0.15,"se":0.02}}
                </div>
                <div id="exposureStatus" class="status" style="display: none;"></div>
            </div>
            <div>
                <label for="outcomeFile">Outcome File</label>
                <div class="file-input-group">
                    <p>Click or drag to upload</p>
                    <input type="file" id="outcomeFile" accept=".csv,.txt,.json,.gz" />
                </div>
                <div class="example-data">
                    <strong>CSV format:</strong> SNP,beta,se,ea,nea,eaf<br>
                    <strong>JSON format:</strong> {"rs9939609": {"beta":-0.08,"se":0.03,"ea":"A","nea":"T"}}
                </div>
                <div id="outcomeStatus" class="status" style="display: none;"></div>
            </div>
        </div>
    </div>
    <!-- Step 2: Harmonization Controls -->
    <div class="form-group">
        <h3 style="color: #667eea; margin-bottom: 15px;">Step 2: Harmonization & Correlation Matrix</h3>
        <div class="correlation-info">
            <strong>Correlation Matrix Source:</strong> Choose between public genetic databases or AI-fetched correlation matrices. AI options (DeepSeek, Gemini) provide alternative correlation estimation. Public databases (NIH LDlink, 1000 Genomes, gnomAD) are free.
        </div>
        <!-- Correlation Source Tabs -->
        <div class="correlation-tabs">
            <button class="correlation-tab active" data-source="ai">AI-Fetched</button>
            <button class="correlation-tab" data-source="public">Public Databases</button>
```

```
                </div>
                <!-- AI-Fetched Content -->
                <div class="correlation-content active" id="ai-correlation">
                    <label for="aiCorrelationService">AI Service for Correlation Matrix</label>
                    <select id="aiCorrelationService">
                        <option value="deepseek">DeepSeek (Recommended)</option>
                        <option value="gemini">Google Gemini</option>
                    </select>
                    <div class="form-group" style="margin-top: 15px;">
                        <label for="aiCorrelationApiKey">API Key</label>
                        <input type="password" id="aiCorrelationApiKey" placeholder="Enter your API key" />
                        <div class="api-info" id="aiCorrelationApiInfo">
                            <strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get API key</a> (Free tier available)<br>
                            <strong>Gemini:</strong> <a href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>
                        </div>
                    </div>
                </div>
                <!-- Public Databases Content -->
                <div class="correlation-content" id="public-correlation">
                    <label for="correlationSource">Correlation Database</label>
                    <select id="correlationSource">
                        <option value="ldlink">NIH LDlink (Public, No Token)</option>
                        <option value="1000genomes">1000 Genomes Project (Public, No Token)</option>
                        <option value="gnomad">gnomAD v3.1 (Optional Token)</option>
                    </select>
                    <div id="tokenGroup" class="form-group" style="display: none; margin-top: 15px;">
                        <label for="dbToken">Database API Token (Optional)</label>
                        <input type="text" id="dbToken" placeholder="Enter API token if available" />
                        <div class="api-info">
                            <strong>gnomAD:</strong> <a href="https://gnomad.broadinstitute.org/" target="_blank">Get token</a>
                        </div>
                    </div>
                </div>
                <button class="btn btn-primary" id="harmonizeBtn" disabled style="margin-top: 15px;">
                    Harmonize Data & Fetch Correlation Matrix
                </button>
            </div>
            <!-- Harmonization Progress -->
            <div id="harmonizationProgress" class="harmonization-progress" style="display: none;">
                <h4 style="margin-bottom: 10px;">Harmonization Progress:</h4>
                <div class="progress-step pending" id="step-load">
                    <span class="step-icon"></span>
```

```
                <span>Loading and parsing files...</span>
            </div>
            <div class="progress-step pending" id="step-harmonize">
                <span class="step-icon"></span>
                <span>Harmonizing exposure and outcome data...</span>
            </div>
            <div class="progress-step pending" id="step-correlation">
                <span class="step-icon"></span>
                <span>Fetching correlation matrix...</span>
            </div>
            <div class="progress-step pending" id="step-json">
                <span class="step-icon"></span>
                <span>Generating final JSON output...</span>
            </div>
        </div>
        <button class="btn btn-secondary back-btn" onclick="backToChoice()">Back to Choices</button>
    </div>
    <!-- Workflow 3: AI and Multi-Source Solutions -->
    <div id="workflow-ai" class="workflow-section">
        <h2>AI and Multi-Source Solutions</h2>
        <p class="subtitle">Choose between AI fetched GWAS data or fetching from multiple databases:</p>
        <!-- Common Analysis Configuration for Both Solutions -->
        <div class="form-group">
            <h3 style="color: #667eea; margin-bottom: 15px;">Step 1: Analysis Configuration</h3>
            <div class="form-group">
                <label>Analysis Type</label>
                <div class="radio-group">
                    <label class="radio-label">
                        <input type="radio" name="analysisType" value="univariate" checked />
                        Univariate MR
                    </label>
                    <label class="radio-label">
                        <input type="radio" name="analysisType" value="multivariate" />
                        Multivariate MR
                    </label>
                </div>
            </div>
            <div class="form-group">
                <label for="exposureTraits">Exposure Trait(s)</label>
                <input type="text" id="exposureTraits" placeholder="e.g., Body Mass Index (BMI) or for multivariate: BMI, HDL Cholesterol, Smoking" />
                <div class="help-text" id="exposureHelpText">For univariate analysis, enter one trait. For multivariate analysis, enter multiple traits separated by commas.</div>
            </div>
            <div class="form-group">
```

```
                    <label for="outcomeVariable">Outcome Variable</label>
                    <input type="text" id="outcomeVariable" placeholder="e.g., Type 2 Diabetes" />
                </div>
            </div>
            <!-- Solution Tabs -->
            <div class="solution-tabs">
                <button class="solution-tab active" data-solution="ai">AI-Powered GWAS Fetching</button>
                <button class="solution-tab" data-solution="gwas">Multi-Source GWAS Fetcher</button>
            </div>
            <!-- AI-Powered GWAS Fetching Content -->
            <div class="solution-content active" id="ai-solution">
                <!-- AI Configuration -->
                <div class="form-group">
                    <h3 style="color: #667eea; margin-bottom: 15px;">Step 2: AI Configuration for Data Fetching</h3>
                    <label for="aiService">AI Service</label>
                    <select id="aiService">
                        <option value="deepseek">DeepSeek (Recommended)</option>
                        <option value="gemini">Google Gemini</option>
                        <option value="gpt">OpenAI GPT</option>
                    </select>
                    <div class="form-group">
                        <label for="apiKey">API Key</label>
                        <input type="password" id="apiKey" placeholder="Enter your API key" />
                        <div class="api-info" id="aiApiInfo">
                            <strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get
API key</a> (Free tier available)<br>
                            <strong>Gemini:</strong>                                                    <a
href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a><br>
                            <strong>GPT:</strong> <a href="https://platform.openai.com/api-keys" target="_blank">Get
API key</a>
                        </div>
                    </div>
                    <button class="btn btn-primary" id="fetchAiDataBtn">Fetch GWAS Data with AI</button>
                </div>
                <!-- AI Loading -->
                <div class="loading" id="aiLoading">
                    <div class="spinner"></div>
                    <p style="color: #667eea; font-weight: 600;">AI is fetching GWAS data... This may take 30-60
seconds.</p>
                </div>
                <!-- AI Fetched Data Preview -->
                <div id="aiDataPreview" style="display: none;">
                    <h3 style="color: #667eea; margin-bottom: 15px;">Step 3: Review AI-Fetched Data</h3>
                    <div class="two-column">
                        <div>
```

```
                        <label>Exposure Data</label>
                        <textarea id="aiExposureData" style="min-height: 250px; font-size: 11px;"></textarea>
                    </div>
                    <div>
                        <label>Outcome Data</label>
                        <textarea id="aiOutcomeData" style="min-height: 250px; font-size: 11px;"></textarea>
                    </div>
                </div>
                <div class="status info" style="margin-top: 15px;">
                    <strong>Review the data above.</strong> You can edit if needed, then proceed to harmonization.
                </div>
                <button class="btn btn-primary" id="harmonizeAiBtn" style="margin-top: 15px;">
                    Harmonize AI Data & Fetch Correlation Matrix
                </button>
            </div>
            <!-- AI Harmonization & Correlation Configuration -->
            <div id="aiHarmonizationConfig" style="display: none;">
                <h3 style="color: #667eea; margin-bottom: 15px;">Step 4: Correlation Matrix Configuration</h3>
                <div class="correlation-info">
                    <strong>Correlation Matrix Source:</strong> Choose between public genetic databases or AI-fetched correlation matrices.
                </div>
                <!-- AI Harmonization Correlation Tabs -->
                <div class="ai-correlation-tabs">
                    <button class="ai-correlation-tab active" data-source="ai">AI-Fetched</button>
                    <button class="ai-correlation-tab" data-source="public">Public Databases</button>
                </div>
                <!-- AI-Fetched for AI Harmonization -->
                <div class="ai-correlation-content active" id="ai-ai-correlation">
                    <label for="aiCorrelationServiceHarmonization">AI Service for Correlation Matrix</label>
                    <select id="aiCorrelationServiceHarmonization">
                        <option value="deepseek">DeepSeek (Recommended - Free Tier)</option>
                        <option value="gemini">Google Gemini</option>
                    </select>
                    <div class="form-group" style="margin-top: 15px;">
                        <label for="aiCorrelationApiKeyHarmonization">API Key</label>
                        <input type="password" id="aiCorrelationApiKeyHarmonization" placeholder="Enter your API key" />
                        <div class="api-info" id="aiCorrelationApiInfoHarmonization">
                            <strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get API key</a> (Free tier available)<br>
                            <strong>Gemini:</strong> <a href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>
                        </div>
                    </div>
```

```
                    </div>
                    <!-- Public Databases for AI Harmonization -->
                    <div class="ai-correlation-content" id="ai-public-correlation">
                        <label for="aiCorrelationSourceDb">Correlation Database</label>
                        <select id="aiCorrelationSourceDb">
                            <option value="ldlink">NIH LDlink (Public, No Token)</option>
                            <option value="1000genomes">1000 Genomes Project (Public, No Token)</option>
                            <option value="gnomad">gnomAD v3.1 (Optional Token)</option>
                        </select>
                        <div id="aiTokenGroup" class="form-group" style="display: none; margin-top: 15px;">
                            <label for="aiDbToken">Database API Token (Optional)</label>
                            <input type="text" id="aiDbToken" placeholder="Enter API token if available" />
                            <div class="api-info">
                                <strong>gnomAD:</strong> <a href="https://gnomad.broadinstitute.org/"
target="_blank">Get token</a>
                            </div>
                        </div>
                    </div>
                </div>
                <!-- AI Harmonization Progress -->
                <div id="aiHarmonizationProgress" class="harmonization-progress" style="display: none;">
                    <h4 style="margin-bottom: 10px;">Harmonization Progress:</h4>
                    <div class="progress-step pending" id="ai-step-parse">
                        <span class="step-icon"></span>
                        <span>Parsing AI-fetched data...</span>
                    </div>
                    <div class="progress-step pending" id="ai-step-harmonize">
                        <span class="step-icon"></span>
                        <span>Harmonizing exposure and outcome data...</span>
                    </div>
                    <div class="progress-step pending" id="ai-step-correlation">
                        <span class="step-icon"></span>
                        <span>Fetching correlation matrix...</span>
                    </div>
                    <div class="progress-step pending" id="ai-step-json">
                        <span class="step-icon"></span>
                        <span>Generating final JSON output...</span>
                    </div>
                </div>
            </div>
            <!-- Multi-Source GWAS Data Fetcher Content -->
            <div class="solution-content" id="gwas-solution">
                <div class="form-group">
                    <h3 style="color: #667eea; margin-bottom: 15px;">Step 2: Data Sources</h3>
                    <div class="info-box">
```

Select one or more databases to search for GWAS summary statistics. Multiple sources increase the chance of finding relevant data.

```
            </div>
            <div class="checkbox-group">
                <div class="checkbox-item">
                    <input type="checkbox" id="source_ieugwas" value="ieugwas" checked>
                    <label for="source_ieugwas">IEU OpenGWAS</label>
                </div>
                <div class="checkbox-item">
                    <input type="checkbox" id="source_gwascatalog" value="gwascatalog" checked>
                    <label for="source_gwascatalog">GWAS Catalog</label>
                </div>
                <div class="checkbox-item">
                    <input type="checkbox" id="source_gwasatlas" value="gwasatlas">
                    <label for="source_gwasatlas">GWAS Atlas</label>
                </div>
                <div class="checkbox-item">
                    <input type="checkbox" id="source_eqtlgen" value="eqtlgen">
                    <label for="source_eqtlgen">eQTLGen</label>
                </div>
                <div class="checkbox-item">
                    <input type="checkbox" id="source_finngen" value="finngen" checked>
                    <label for="source_finngen">FinnGen</label>
                </div>
            </div>
        </div>
        <div class="form-group">
            <h3 style="color: #667eea; margin-bottom: 15px;">Step 3: API Authentication (Optional)</h3>
            <div class="info-box">
                Some databases may require authentication tokens for access or higher rate limits. Leave blank if not required.
            </div>
            <div class="api-token-group">
                <div class="api-token-item">
                    <label for="apiToken_ieugwas">IEU OpenGWAS Token:</label>
                    <input type="text" id="apiToken_ieugwas" placeholder="Optional">
                </div>
                <div class="api-token-item">
                    <label for="apiToken_gwascatalog">GWAS Catalog Token:</label>
                    <input type="text" id="apiToken_gwascatalog" placeholder="Optional">
                </div>
                <div class="api-token-item">
                    <label for="apiToken_gwasatlas">GWAS Atlas Token:</label>
                    <input type="text" id="apiToken_gwasatlas" placeholder="Optional">
                </div>
```

```
                        <div class="api-token-item">
                            <label for="apiToken_eqtlgen">eQTLGen Token:</label>
                            <input type="text" id="apiToken_eqtlgen" placeholder="Optional">
                        </div>
                        <div class="api-token-item">
                            <label for="apiToken_finngen">FinnGen Token:</label>
                            <input type="text" id="apiToken_finngen" placeholder="Optional">
                        </div>
                    </div>
                </div>
                <div class="form-group">
                    <label for="gwasPvalThreshold">P-value Threshold:</label>
                    <input type="number" id="gwasPvalThreshold" value="5e-8" step="1e-9" min="0" max="1">
                    <div class="help-text">SNPs with p-values below this threshold will be considered genome-wide significant.</div>
                </div>
                <button class="btn btn-primary" id="fetchGwasButton">Fetch GWAS Data</button>
                <div id="gwasProgressContainer" style="display: none; margin-top: 20px;">
                    <div id="progressBar">
                        <div id="progressFill">0%</div>
                    </div>
                    <div id="gwasStatus" class="status info" style="margin-top: 8px;">Initializing...</div>
                </div>
                <div id="gwasResults" style="margin-top: 20px;"></div>
                <!-- GWAS Harmonization Configuration -->
                <div id="gwasHarmonizationSection" style="display: none; margin-top: 30px;">
                    <h3 style="color: #667eea; margin-bottom: 15px;">Step 4: Harmonize Fetched Data</h3>
                    <div class="correlation-info">
                        <strong>Correlation Matrix Source:</strong> Choose between public genetic databases or AI-fetched correlation matrices.
                    </div>
                    <!-- GWAS Harmonization Correlation Tabs -->
                    <div class="correlation-tabs">
                        <button class="correlation-tab active" data-source="ai" id="gwas-ai-tab">AI-Fetchted</button>
                        <button class="correlation-tab" data-source="public" id="gwas-public-tab">Public Databases</button>
                    </div>
                    <!-- AI-Fetched for GWAS Harmonization -->
                    <div class="correlation-content active" id="gwas-ai-correlation">
                        <label for="gwasAiCorrelationService">AI Service for Correlation Matrix</label>
                        <select id="gwasAiCorrelationService">
                            <option value="deepseek">DeepSeek (Recommended)</option>
                            <option value="gemini">Google Gemini</option>
                        </select>
                        <div class="form-group" style="margin-top: 15px;">
```

```
                                <label for="gwasAiCorrelationApiKey">API Key</label>
                                <input type="password" id="gwasAiCorrelationApiKey" placeholder="Enter your API key"
/>
                                <div class="api-info">
                                    <strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/"
target="_blank">Get API key</a> (Free tier available)<br>
                                    <strong>Gemini:</strong> <a
href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>
                                </div>
                            </div>
                        </div>
                        <!-- Public Databases for GWAS Harmonization -->
                        <div class="correlation-content" id="gwas-public-correlation">
                            <label for="gwasCorrelationSourceDb">Correlation Database</label>
                            <select id="gwasCorrelationSourceDb">
                                <option value="ldlink">NIH LDlink (Public, No Token)</option>
                                <option value="1000genomes">1000 Genomes Project (Public, No Token)</option>
                                <option value="gnomad">gnomAD v3.1 (Optional Token)</option>
                            </select>
                            <div id="gwasTokenGroup" class="form-group" style="display: none; margin-top: 15px;">
                                <label for="gwasDbToken">Database API Token (Optional)</label>
                                <input type="text" id="gwasDbToken" placeholder="Enter API token if available" />
                                <div class="api-info">
                                    <strong>gnomAD:</strong> <a href="https://gnomad.broadinstitute.org/"
target="_blank">Get token</a>
                                </div>
                            </div>
                        </div>
                        <button class="btn btn-primary" id="harmonizeGwasBtn" style="margin-top: 15px;">
                            Harmonize GWAS Data & Generate JSON
                        </button>
                    </div>
                    <!-- GWAS Harmonization Progress -->
                    <div id="gwasHarmonizationProgress" class="harmonization-progress" style="display: none;">
                        <h4 style="margin-bottom: 10px;">Harmonization Progress:</h4>
                        <div class="progress-step pending" id="gwas-step-parse">
                            <span class="step-icon"></span>
                            <span>Parsing GWAS data...</span>
                        </div>
                        <div class="progress-step pending" id="gwas-step-harmonize">
                            <span class="step-icon"></span>
                            <span>Harmonizing exposure and outcome data...</span>
                        </div>
                        <div class="progress-step pending" id="gwas-step-correlation">
                            <span class="step-icon"></span>
```

```
                    <span>Fetching correlation matrix...</span>
                </div>
                <div class="progress-step pending" id="gwas-step-json">
                    <span class="step-icon"></span>
                    <span>Generating final JSON output...</span>
                </div>
            </div>
        </div>
        <button class="btn btn-secondary back-btn" onclick="backToChoice()">Back to Choices</button>
    </div>
    <!-- Global JSON Output Section -->
    <div class="form-group" id="globalOutput" style="display: none; margin-top: 30px;">
        <h2 style="color: #667eea; margin-bottom: 15px;">Final JSON Output</h2>
        <textarea id="jsonoutput" class="output-json" readonly placeholder="Your generated JSON will appear here..."></textarea>
        <div style="margin-top: 15px;">
            <button class="btn btn-success" id="copyJsonOutputBtn">Copy JSON</button>
            <button class="btn btn-success" id="downloadJsonOutputBtn">Download JSON</button>
        </div>
    </div>
</div>
<!-- Dependencies -->
<script src="http://www.iaees.org/publications/software/JavaScript/pako.min.js"></script>
<script>
    // ============ Enhanced Classes from file 33.txt ============
    // 1. Rate Limiter Class
    class RateLimiter {
        constructor(maxRequests = 10, perSeconds = 1) {
            this.maxRequests = maxRequests;
            this.perSeconds = perSeconds;
            this.requests = [];
        }
        async waitForSlot() {
            const now = Date.now();
            const cutoff = now - (this.perSeconds * 1000);
            // Remove old requests
            this.requests = this.requests.filter(time => time > cutoff);
            if (this.requests.length >= this.maxRequests) {
                const oldestRequest = this.requests[0];
                const waitTime = (oldestRequest + (this.perSeconds * 1000)) - now;
                await new Promise(resolve => setTimeout(resolve, waitTime));
                return this.waitForSlot();
            }
            this.requests.push(now);
        }
```

```
}
// 2. Data Cache Class
class DataCache {
    constructor(maxAge = 3600000) { // 1 hour default
        this.cache = new Map();
        this.maxAge = maxAge;
    }
    set(key, value) {
        this.cache.set(key, {
            data: value,
            timestamp: Date.now()
        });
    }
    get(key) {
        const entry = this.cache.get(key);
        if (!entry) return null;
        if (Date.now() - entry.timestamp > this.maxAge) {
            this.cache.delete(key);
            return null;
        }
        return entry.data;
    }
    clear() {
        this.cache.clear();
    }
}
// 3. Loading Manager Class
class LoadingManager {
    constructor() {
        this.activeOperations = new Set();
    }
    start(operationId, message) {
        this.activeOperations.add(operationId);
        this.updateUI(operationId, 'loading', message);
    }
    update(operationId, message) {
        if (this.activeOperations.has(operationId)) {
            this.updateUI(operationId, 'loading', message);
        }
    }
    complete(operationId, message) {
        this.activeOperations.delete(operationId);
        this.updateUI(operationId, 'complete', message);
    }
    error(operationId, message) {
```

```
                this.activeOperations.delete(operationId);
                this.updateUI(operationId, 'error', message);
            }
            updateUI(operationId, status, message) {
                const statusEl = document.getElementById(`status_${operationId}`);
                if (statusEl) {
                    statusEl.textContent = message;
                    statusEl.className = `status ${status}`;
                }
            }
        }
        // Initialize enhanced classes
        const apiLimiters = {
            ieugwas: new RateLimiter(10, 1),
            gwascatalog: new RateLimiter(5, 1),
            finngen: new RateLimiter(10, 1),
            gwasatlas: new RateLimiter(5, 1),
            eqtlgen: new RateLimiter(5, 1)
        };
        const gwasCache = new DataCache();
        const loadingManager = new LoadingManager();
        // CORS Proxy Configuration
        const USE_PROXY = false; // Set to true if using a backend proxy
        const PROXY_URL = 'https://iaees.net/api/proxy/'; // Configure proxy URL
        // ============ Enhanced Utility Functions ============
        // Enhanced CORS-aware fetch
        async function fetchWithCORS(url, options = {}) {
            const finalUrl = USE_PROXY ? PROXY_URL + encodeURIComponent(url) : url;
            try {
                const response = await fetch(finalUrl, options);
                if (!response.ok) {
                    throw new Error(`HTTP error! status: ${response.status}`);
                }
                return response;
            } catch (error) {
                if (error.message.includes('CORS') || error.message.includes('NetworkError')) {
                    console.warn('CORS/Network error detected. Consider using a proxy server or backend service.');
                    throw new Error(`Network/CORS error: ${error.message}. Try using a backend proxy for production
use.`);
                }
                throw error;
            }
        }
        // Enhanced SNP Data Validation
        function validateSNPData(snp, source) {
```

```
        const issues = [];
        if (!snp.rsid || !snp.rsid.match(/^rs\d+$/)) {
            issues.push('Invalid or missing rsID');
        }
        if (snp.beta === null || snp.beta === undefined || isNaN(snp.beta)) {
            issues.push('Invalid beta value');
        }
        if (snp.se === null || snp.se === undefined || isNaN(snp.se) || snp.se <= 0) {
            issues.push('Invalid standard error');
        }
        if (snp.p !== null && snp.p !== undefined) {
            if (isNaN(snp.p) || snp.p < 0 || snp.p > 1) {
                issues.push('Invalid p-value');
            }
        }
        if (snp.eaf !== null && snp.eaf !== undefined) {
            if (isNaN(snp.eaf) || snp.eaf < 0 || snp.eaf > 1) {
                issues.push('Invalid allele frequency');
            }
        }
        if (snp.ea && !['A', 'T', 'C', 'G'].includes(snp.ea.toUpperCase())) {
            issues.push('Invalid effect allele');
        }
        if (snp.nea && !['A', 'T', 'C', 'G'].includes(snp.nea.toUpperCase())) {
            issues.push('Invalid non-effect allele');
        }
        if (issues.length > 0) {
            console.warn(`SNP ${snp.rsid || 'unknown'} from ${source}: ${issues.join(', ')}`);
            return {
                valid: false,
                issues: issues,
                snp: snp
            };
        }
        return {
            valid: true,
            snp: snp
        };
    }
    // Enhanced error handling with retry logic
    async function fetchWithRetry(url, options = {}, maxRetries = 3, baseDelay = 1000) {
        let lastError;
        for (let attempt = 1; attempt <= maxRetries; attempt++) {
            try {
                const response = await fetchWithCORS(url, options);
```

```
                if (!response.ok) {
                    throw new Error(`HTTP ${response.status}: ${response.statusText}`);
                }
                return response;
            } catch (error) {
                lastError = error;
                console.warn(`Attempt ${attempt} failed for ${url}: ${error.message}`);
                if (attempt < maxRetries) {
                    // Exponential backoff with jitter
                    const delay = baseDelay * Math.pow(2, attempt - 1) + Math.random() * 100;
                    console.log(`Retrying in ${Math.round(delay)}ms...`);
                    await new Promise(resolve => setTimeout(resolve, delay));
                }
            }
        }
        throw new Error(`Failed after ${maxRetries} attempts: ${lastError.message}`);
    }
    // ============ Global State ============
    const STATE = {
        workflow: null,
        exposureData: {},
        outcomeData: {},
        harmonizedData: null,
        correlationMatrix: null,
        finalJson: null,
        correlationSource: 'ai',
        aiCorrelationSource: 'ai',
        gwasCorrelationSource: 'ai',
        gwasRawData: {
            exposure: {},
            outcome: {}
        }
    };
    // ============ Workflow Selection ============
    document.querySelectorAll('.choice-card').forEach(card => {
        card.addEventListener('click', () => {
            const workflow = card.dataset.workflow;
            selectWorkflow(workflow);
        });
    });
    function selectWorkflow(workflow) {
        STATE.workflow = workflow;
        // Hide choice section
        document.getElementById('workflowChoice').style.display = 'none';
        // Show selected workflow
```

```
        document.querySelectorAll('.workflow-section').forEach(section => {
            section.classList.remove('active');
        });
        document.getElementById(`workflow-${workflow}`).classList.add('active');
        // Initialize workflow-specific features
        if (workflow === 'ai') {
            initializeAIWorkflow();
        } else if (workflow === 'files') {
            initializeFilesWorkflow();
        }
    }
    function backToChoice() {
        document.getElementById('workflowChoice').style.display = 'block';
        document.querySelectorAll('.workflow-section').forEach(section => {
            section.classList.remove('active');
        });
        // Reset state
        STATE.workflow = null;
        STATE.exposureData = {};
        STATE.outcomeData = {};
        STATE.harmonizedData = null;
        STATE.correlationMatrix = null;
        STATE.finalJson = null;
        STATE.correlationSource = 'ai';
        STATE.aiCorrelationSource = 'ai';
        STATE.gwasCorrelationSource = 'ai';
        STATE.gwasRawData = { exposure: {}, outcome: {} };
        // Clear cache for fresh start
        gwasCache.clear();
    }
    // ============ Workflow 1: Direct JSON Input ============
    document.getElementById('validateJsonBtn').addEventListener('click', validateDirectJson);
    document.getElementById('copyJsonOutputBtn').addEventListener('click', () => copyToClipboard('jsonoutput'));
    document.getElementById('downloadJsonOutputBtn').addEventListener('click', () =>
downloadJson(document.getElementById('jsonoutput').value, 'mr_data.json'));
    function validateDirectJson() {
        const textarea = document.getElementById('jsonDirectInput');
        const status = document.getElementById('jsonStatus');
        try {
            const jsonText = textarea.value.trim();
            if (!jsonText) {
                throw new Error('Please paste JSON data first');
            }
            const parsed = JSON.parse(jsonText);
            // Validate structure
```

```
            const required = ['snps', 'betaX', 'betaY', 'betaXse', 'betaYse'];
            for (const field of required) {
                if (!parsed[field] || !Array.isArray(parsed[field])) {
                    throw new Error(`Missing or invalid field: ${field}`);
                }
            }
            // Validate array lengths
            const length = parsed.snps.length;
            if (parsed.betaX.length !== length || parsed.betaY.length !== length ||
                parsed.betaXse.length !== length || parsed.betaYse.length !== length) {
                throw new Error('All arrays must have the same length');
            }
            // Validate correlation matrix if present
            if (parsed.correlation) {
                if (!Array.isArray(parsed.correlation) || parsed.correlation.length !== length) {
                    throw new Error('Correlation matrix must be a 2D array matching SNP count');
                }
                for (let i = 0; i < length; i++) {
                    if (!Array.isArray(parsed.correlation[i]) || parsed.correlation[i].length !== length) {
                        throw new Error(`Correlation matrix row ${i} has incorrect length`);
                    }
                }
            }
            // Pretty print in the original textarea
            textarea.value = JSON.stringify(parsed, null, 2);
            // Output validated JSON to the global jsonoutput textarea
            document.getElementById('jsonoutput').value = JSON.stringify(parsed, null, 2);
            document.getElementById('globalOutput').style.display = 'block';
            status.textContent = `Valid JSON! ${length} SNPs validated successfully.`;
            status.className = 'status success';
            status.style.display = 'block';
        } catch (error) {
            status.textContent = `Invalid JSON: ${error.message}`;
            status.className = 'status error';
            status.style.display = 'block';
        }
    }
    // ============ Workflow 2: File Upload & Harmonization ============
    function initializeFilesWorkflow() {
        const correlationSource = document.getElementById('correlationSource');
        const tokenGroup = document.getElementById('tokenGroup');
        const correlationTabs = document.querySelectorAll('.correlation-tab');
        const aiCorrelationService = document.getElementById('aiCorrelationService');
        // Correlation source database change
        correlationSource.addEventListener('change', () => {
```

```
            if (correlationSource.value === 'ldlink' || correlationSource.value === '1000genomes') {
                tokenGroup.style.display = 'none';
            } else {
                tokenGroup.style.display = 'block';
            }
        });
        // Correlation source tab switching
        correlationTabs.forEach(tab => {
            tab.addEventListener('click', () => {
                const source = tab.dataset.source;
                STATE.correlationSource = source;
                // Update tab active state
                correlationTabs.forEach(t => t.classList.remove('active'));
                tab.classList.add('active');
                // Update content visibility
                document.querySelectorAll('.correlation-content').forEach(content => {
                    content.classList.remove('active');
                });
                document.getElementById(`${source}-correlation`).classList.add('active');
            });
        });
        // AI correlation service change
        aiCorrelationService.addEventListener('change', updateAICorrelationInfo);
        updateAICorrelationInfo(); // Initial call to set info
    }
    function updateAICorrelationInfo() {
        const service = document.getElementById('aiCorrelationService').value;
        const info = document.getElementById('aiCorrelationApiInfo');
        const infos = {
            deepseek: '<strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get API
key</a> (Free tier available)',
            gemini:                                    '<strong>Gemini:</strong>                                    <a
href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>'
        };
        info.innerHTML = infos[service];
    }
    document.getElementById('exposureFile').addEventListener('change', handleExposureFile);
    document.getElementById('outcomeFile').addEventListener('change', handleOutcomeFile);
    document.getElementById('harmonizeBtn').addEventListener('click', harmonizeFiles);
    document.getElementById('copyJsonOutputBtn').addEventListener('click', () => copyToClipboard('jsonoutput'));
    document.getElementById('downloadJsonOutputBtn').addEventListener('click',                    ()                    =>
downloadJson(document.getElementById('jsonoutput').value, 'mr_harmonized_data.json'));
    async function handleExposureFile(e) {
        const file = e.target.files[0];
        if (!file) return;
```

```
        const status = document.getElementById('exposureStatus');
        status.textContent = `Loading ${file.name}...`;
        status.className = 'status info';
        status.style.display = 'block';
        try {
            const data = await parseFile(file);
            STATE.exposureData = data;
            status.textContent = `Loaded ${Object.keys(data).length} exposure variants from ${file.name}`;
            status.className = 'status success';
            checkFilesReady();
        } catch (error) {
            status.textContent = `Error: ${error.message}`;
            status.className = 'status error';
        }
    }
    async function handleOutcomeFile(e) {
        const file = e.target.files[0];
        if (!file) return;
        const status = document.getElementById('outcomeStatus');
        status.textContent = `Loading ${file.name}...`;
        status.className = 'status info';
        status.style.display = 'block';
        try {
            const data = await parseFile(file);
            STATE.outcomeData = data;
            status.textContent = `Loaded ${Object.keys(data).length} outcome variants from ${file.name}`;
            status.className = 'status success';
            checkFilesReady();
        } catch (error) {
            status.textContent = `Error: ${error.message}`;
            status.className = 'status error';
        }
    }
    function checkFilesReady() {
        const harmonizeBtn = document.getElementById('harmonizeBtn');
        if (Object.keys(STATE.exposureData).length > 0 && Object.keys(STATE.outcomeData).length > 0) {
            harmonizeBtn.disabled = false;
        }
    }
    async function parseFile(file) {
        const buf = await file.arrayBuffer();
        let text;
        // Handle gzip
        if (file.name.endsWith('.gz') || (new Uint8Array(buf, 0, 2)[0] === 0x1f && new Uint8Array(buf, 0, 2)[1] ===
0x8b)) {
```

```
            text = pako.ungzip(new Uint8Array(buf), { to: 'string' });
        } else {
            text = new TextDecoder().decode(buf);
        }
        // Try JSON first
        if (file.name.endsWith('.json') || text.trim().startsWith('{')) {
            try {
                const parsed = JSON.parse(text);
                return normalizeJsonData(parsed);
            } catch (e) {
                // Not JSON, fall through to CSV
            }
        }
        // Parse CSV/TSV
        return parseCsvData(text);
    }
    function normalizeJsonData(parsed) {
        const result = {};
        if (Array.isArray(parsed)) {
            parsed.forEach(row => {
                const normalized = normalizeRow(row);
                if (normalized.rsid) {
                    result[normalized.rsid] = normalized;
                }
            });
        } else if (typeof parsed === 'object') {
            Object.entries(parsed).forEach(([key, value]) => {
                const normalized = normalizeRow({ ...value, rsid: key });
                if (normalized.rsid) {
                    result[normalized.rsid] = normalized;
                }
            });
        }
        return result;
    }
    function parseCsvData(text) {
        const lines = text.split(/\r?\n/).filter(l => l.trim());
        if (lines.length < 2) throw new Error('File is empty or has no data rows');
        const firstLine = lines[0];
        const delim = firstLine.includes('\t') ? '\t' : ',';
        const headers = firstLine.split(delim).map(h => h.trim().replace(/^"|"$/g, '').toLowerCase());
        const result = {};
        for (let i = 1; i < lines.length; i++) {
            const parts = lines[i].split(delim).map(p => p.trim().replace(/^"|"$/g, ''));
            const obj = {};
```

```
            headers.forEach((header, idx) => {
                if (idx < parts.length) {
                    obj[header] = parts[idx];
                }
            });
            const normalized = normalizeRow(obj);
            if (normalized.rsid) {
                result[normalized.rsid] = normalized;
            }
        }
        return result;
    }
    function normalizeRow(row) {
        const fieldMap = {
            rsid: ['snp', 'rsid', 'variant', 'markername', 'id', 'snpid'],
            ea: ['effect_allele', 'ea', 'a1', 'effectallele', 'allele1'],
            nea: ['other_allele', 'nea', 'a2', 'otherallele', 'allele2'],
            beta: ['beta', 'b', 'beta_estimate', 'effect_size', 'betavalue'],
            se: ['se', 'stderr', 'beta_se', 'betavaluese'],
            eaf: ['eaf', 'maf', 'allele_frequency', 'af', 'frequency']
        };
        const lowerRow = {};
        Object.entries(row).forEach(([k, v]) => {
            lowerRow[k.toLowerCase()] = v;
        });
        const result = {};
        Object.entries(fieldMap).forEach(([target, keys]) => {
            for (const key of keys) {
                if (key in lowerRow && lowerRow[key] != null && lowerRow[key] !== '') {
                    result[target] = lowerRow[key];
                    break;
                }
            }
        });
        // Convert numbers
        if (result.beta) result.beta = Number(result.beta);
        if (result.se) result.se = Number(result.se);
        if (result.eaf) result.eaf = Number(result.eaf);
        return result;
    }
    async function harmonizeFiles() {
        const progress = document.getElementById('harmonizationProgress');
        progress.style.display = 'block';
        try {
            // Step 1: Already loaded
```

```
        updateProgressStep('step-load', 'complete', 'Files loaded successfully');
        // Step 2: Harmonize
        updateProgressStep('step-harmonize', 'active', 'Harmonizing data...');
        await new Promise(resolve => setTimeout(resolve, 500));
        const harmonized = harmonizeData(STATE.exposureData, STATE.outcomeData);
        STATE.harmonizedData = harmonized;
        updateProgressStep('step-harmonize', 'complete', `Harmonized ${harmonized.snps.length} common SNPs`);
        // Step 3: Fetch correlation
        updateProgressStep('step-correlation', 'active', 'Fetching correlation matrix...');
        let correlation;
        if (STATE.correlationSource === 'ai') {
            const service = document.getElementById('aiCorrelationService').value;
            const apiKey = document.getElementById('aiCorrelationApiKey').value.trim();
            if (!apiKey) {
                throw new Error('Please provide an API key for AI correlation matrix fetching');
            }
            correlation = await fetchAICorrelationMatrix(harmonized.snps, service, apiKey);
        } else {
            const source = document.getElementById('correlationSource').value;
            const token = document.getElementById('dbToken').value.trim();
            correlation = await fetchCorrelationMatrix(harmonized.snps, source, token);
        }
        STATE.correlationMatrix = correlation;
        if (correlation.source === 'identity') {
            updateProgressStep('step-correlation', 'complete', 'Database unavailable - using identity matrix');
        } else {
            updateProgressStep('step-correlation', 'complete', `Correlation matrix fetched from
${correlation.source}`);
        }
        // Step 4: Generate JSON
        updateProgressStep('step-json', 'active', 'Generating final JSON...');
        await new Promise(resolve => setTimeout(resolve, 300));
        const finalJson = {
            snps: harmonized.snps,
            betaX: harmonized.betaX,
            betaY: harmonized.betaY,
            betaXse: harmonized.betaXse,
            betaYse: harmonized.betaYse,
            correlation: correlation.matrix,
            exposure: "Exposure",
            outcome: "Outcome",
            n: harmonized.snps.length,
            metadata: {
                harmonized_snps: harmonized.snps.length,
                correlation_source: correlation.source,
```

```
                correlation_method: correlation.method,
                timestamp: new Date().toISOString()
            }
        };
        STATE.finalJson = finalJson;
        updateProgressStep('step-json', 'complete', 'JSON generated successfully');
        // Display output in global textarea
        document.getElementById('jsonoutput').value = JSON.stringify(finalJson, null, 2);
        document.getElementById('globalOutput').style.display = 'block';
    } catch (error) {
        console.error('Harmonization error:', error);
        const activeStep = document.querySelector('.progress-step.active');
        if (activeStep) {
            updateProgressStep(activeStep.id, 'error', `Error: ${error.message}`);
        }
    }
}
function harmonizeData(exposureData, outcomeData) {
    const commonRsids = Object.keys(exposureData).filter(rsid => outcomeData[rsid]);
    const snps = [];
    const betaX = [];
    const betaY = [];
    const betaXse = [];
    const betaYse = [];
    for (const rsid of commonRsids) {
        const exp = exposureData[rsid];
        const out = outcomeData[rsid];
        // Simple allele alignment
        let betaYValue = out.beta;
        // Flip if alleles are reversed
        if (exp.ea && exp.nea && out.ea && out.nea) {
            if (exp.ea === out.nea && exp.nea === out.ea) {
                betaYValue = -betaYValue;
            }
        }
        snps.push(rsid);
        betaX.push(exp.beta);
        betaY.push(betaYValue);
        betaXse.push(exp.se);
        betaYse.push(out.se);
    }
    return { snps, betaX, betaY, betaXse, betaYse };
}
async function fetchCorrelationMatrix(snps, source, token) {
    try {
```

```
            // Try to fetch from actual database
            if (source === 'ldlink') {
                return await fetchLDlinkCorrelation(snps);
            } else if (source === '1000genomes') {
                return await fetch1000GenomesCorrelation(snps);
            } else if (source === 'gnomad') {
                return await fetchGnomADCorrelation(snps, token);
            }
        } catch (error) {
            console.warn(`Failed to fetch from ${source}:`, error);
        }
        // Fallback to identity matrix
        return createIdentityMatrix(snps);
    }
    async function fetchLDlinkCorrelation(snps) {
        // Use NIH LDlink API with enhanced error handling
        const snpSubset = snps.slice(0, 10); // Limit for API
        try {
            const snpString = snpSubset.join('%0A');
            const url =
`https://ldlink.nci.nih.gov/LDlinkRest/ldmatrix?snps=${snpString}&pop=CEU&r2_d=r2&token=dummy`;
            const response = await fetchWithRetry(url, {
                headers: { 'Accept': 'text/plain' }
            });
            const text = await response.text();
            const matrix = parseLDMatrix(text, snpSubset.length);
            // Extend matrix to full size with identity for remaining SNPs
            const fullMatrix = extendMatrixToFullSize(matrix, snps.length);
            return {
                matrix: fullMatrix,
                source: 'ldlink',
                method: 'ldlink_api'
            };
        } catch (error) {
            console.warn('LDlink fetch failed:', error);
            throw new Error('LDlink fetch failed - using fallback matrix');
        }
    }
    async function fetch1000GenomesCorrelation(snps) {
        // Enhanced 1000 Genomes implementation would go here
        throw new Error('1000 Genomes API not available in this version');
    }
    async function fetchGnomADCorrelation(snps, token) {
        // Enhanced gnomAD implementation would go here
        throw new Error('gnomAD API not available in this version');
```

```
        }
        async function fetchAICorrelationMatrix(snps, service, apiKey) {
            const snpList = snps.join(', ');
            const prompt = `You are a genetics expert. For the following SNPs, fetch a realistic genetic correlation matrix based on typical linkage disequilibrium patterns.
SNPs: ${snpList}
Generate a JSON response with ONLY the following format (no markdown, no extra text):
{
  "matrix": [[1.0, 0.15, 0.08, ...], [0.15, 1.0, 0.12, ...], ...],
  "description": "LD-based correlation matrix"
}
Rules:
1. Diagonal elements must be 1.0
2. Matrix must be symmetric
3. All values between -1 and 1
4. Use realistic LD patterns (R¬≤ values converted to correlation)
5. Nearby SNPs should have higher correlation`;
            try {
                if (service === 'deepseek') {
                    const response = await fetchWithRetry('https://api.deepseek.com/chat/completions', {
                        method: 'POST',
                        headers: {
                            'Content-Type': 'application/json',
                            'Authorization': `Bearer ${apiKey}`
                        },
                        body: JSON.stringify({
                            model: 'deepseek-chat',
                            messages: [
                                { role: 'system', content: 'You are a genetics data expert. Respond with valid JSON only.' },
                                { role: 'user', content: prompt }
                            ],
                            temperature: 0.5,
                            max_tokens: 2000
                        })
                    });
                    if (!response.ok) {
                        const error = await response.json();
                        throw new Error(error.error?.message || `API error: ${response.status}`);
                    }
                    const data = await response.json();
                    const content = data.choices?.[0]?.message?.content;
                    if (!content) {
                        throw new Error('No content received from AI');
                    }
```

```
                return parseAICorrelationResponse(content, snps);
            } else if (service === 'gemini') {
                const response = await fetchWithRetry(`https://generativelanguage.googleapis.com/v1/models/gemini-1.5-flash:generateContent?key=${apiKey}`, {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({
                        contents: [{ parts: [{ text: prompt }] }],
                        generationConfig: { temperature: 0.5, maxOutputTokens: 2000 }
                    })
                });
                if (!response.ok) {
                    const error = await response.json();
                    throw new Error(error.error?.message || `API error: ${response.status}`);
                }
                const data = await response.json();
                const content = data.candidates?.[0]?.content?.parts?.[0]?.text;
                if (!content) {
                    throw new Error('No content received from AI');
                }
                return parseAICorrelationResponse(content, snps);
            }
        } catch (error) {
            console.warn(`AI correlation fetch failed: ${error.message}`);
            throw new Error(`AI correlation matrix fetching failed: ${error.message}`);
        }
    }
    function parseAICorrelationResponse(content, snps) {
        try {
            // Extract JSON from response
            const jsonMatch = content.match(/\{[\s\S]*\}/);
            if (!jsonMatch) {
                throw new Error('No JSON found in response');
            }
            const parsed = JSON.parse(jsonMatch[0]);
            if (!parsed.matrix || !Array.isArray(parsed.matrix)) {
                throw new Error('Invalid matrix format in response');
            }
            // Validate matrix dimensions
            if (parsed.matrix.length !== snps.length) {
                throw new Error(`Matrix dimension mismatch: expected ${snps.length}, got ${parsed.matrix.length}`);
            }
            for (let i = 0; i < parsed.matrix.length; i++) {
                if (!Array.isArray(parsed.matrix[i]) || parsed.matrix[i].length !== snps.length) {
                    throw new Error(`Row ${i} has incorrect length`);
```

```
                }
            }
            return {
                matrix: parsed.matrix,
                source: 'ai_deepseek' || 'ai_gemini',
                method: 'ld_based_ai_estimation'
            };
        } catch (error) {
            throw new Error(`Failed to parse AI correlation response: ${error.message}`);
        }
    }
    function parseLDMatrix(text, size) {
        const lines = text.trim().split('\n');
        const matrix = [];
        for (let i = 1; i <= size && i < lines.length; i++) {
            const parts = lines[i].split('\t');
            const row = parts.slice(1, size + 1).map(v => {
                const num = parseFloat(v);
                return isNaN(num) ? 0 : Math.sqrt(Math.max(0, num)); // R¬≤ to r
            });
            matrix.push(row);
        }
        return matrix;
    }
    function extendMatrixToFullSize(partialMatrix, fullSize) {
        const matrix = [];
        for (let i = 0; i < fullSize; i++) {
            const row = [];
            for (let j = 0; j < fullSize; j++) {
                if (i < partialMatrix.length && j < partialMatrix[i].length) {
                    row.push(partialMatrix[i][j]);
                } else {
                    row.push(i === j ? 1.0 : 0.0);
                }
            }
            matrix.push(row);
        }
        return matrix;
    }
    function createIdentityMatrix(snps) {
        const n = snps.length;
        const matrix = [];
        for (let i = 0; i < n; i++) {
            const row = [];
            for (let j = 0; j < n; j++) {
```

```
                row.push(i === j ? 1.0 : 0.0);
            }
            matrix.push(row);
        }
        return {
            matrix: matrix,
            source: 'identity',
            method: 'fallback'
        };
    }
    function updateProgressStep(stepId, status, text) {
        const step = document.getElementById(stepId);
        if (!step) return;
        step.className = `progress-step ${status}`;
        const icons = {
            pending: '',
            active: '',
            complete: '◎',
            error: '▽'
        };
        step.querySelector('.step-icon').textContent = icons[status] || '';
        step.querySelector('span:last-child').textContent = text;
    }
    // ============ Workflow 3: AI and Multi-Source Solutions ============
    function initializeAIWorkflow() {
        // Initialize common analysis type radio buttons
        const analysisTypeRadios = document.querySelectorAll('input[name="analysisType"]');
        const exposureHelpText = document.getElementById('exposureHelpText');
        // Update help text based on analysis type
        analysisTypeRadios.forEach(radio => {
            radio.addEventListener('change', function() {
                if (this.value === 'multivariate') {
                    exposureHelpText.textContent = 'For multivariate analysis, enter multiple traits separated by commas (e.g., BMI, HDL Cholesterol, Smoking).';
                } else {
                    exposureHelpText.textContent = 'For univariate analysis, enter one trait.';
                }
            });
        });
        // Initialize solution tabs
        const solutionTabs = document.querySelectorAll('.solution-tab');
        solutionTabs.forEach(tab => {
            tab.addEventListener('click', () => {
                const solution = tab.dataset.solution;
                // Update tab active state
```

```
                solutionTabs.forEach(t => t.classList.remove('active'));
                tab.classList.add('active');
                // Update content visibility
                document.querySelectorAll('.solution-content').forEach(content => {
                    content.classList.remove('active');
                });
                document.getElementById(`${solution}-solution`).classList.add('active');
                // Initialize specific solution if needed
                if (solution === 'ai') {
                    initializeAISolution();
                } else if (solution === 'gwas') {
                    initializeGWASSolution();
                }
            });
        });
        // Initialize AI solution by default
        initializeAISolution();
    }
    function initializeAISolution() {
        const aiService = document.getElementById('aiService');
        aiService.addEventListener('change', updateAIServiceInfo);
        updateAIServiceInfo(); // Initial call to set info
        // Initialize AI harmonization correlation tabs
        const aiCorrelationTabs = document.querySelectorAll('.ai-correlation-tab');
        const aiCorrelationSourceDb = document.getElementById('aiCorrelationSourceDb');
        const aiTokenGroup = document.getElementById('aiTokenGroup');
        const aiCorrelationServiceHarmonization = document.getElementById('aiCorrelationServiceHarmonization');
        aiCorrelationTabs.forEach(tab => {
            tab.addEventListener('click', () => {
                const source = tab.dataset.source;
                STATE.aiCorrelationSource = source;
                // Update tab active state
                aiCorrelationTabs.forEach(t => t.classList.remove('active'));
                tab.classList.add('active');
                // Update content visibility
                document.querySelectorAll('.ai-correlation-content').forEach(content => {
                    content.classList.remove('active');
                });
                document.getElementById(`ai-${source}-correlation`).classList.add('active');
            });
        });
        aiCorrelationSourceDb.addEventListener('change', () => {
            if (aiCorrelationSourceDb.value === 'ldlink' || aiCorrelationSourceDb.value === '1000genomes') {
                aiTokenGroup.style.display = 'none';
            } else {
```

```
                aiTokenGroup.style.display = 'block';
            }
        });
        aiCorrelationServiceHarmonization.addEventListener('change', updateAICorrelationInfoHarmonization);
        updateAICorrelationInfoHarmonization(); // Initial call to set info
    }
    function initializeGWASSolution() {
        // Initialize correlation tabs for GWAS solution
        const gwasCorrelationTabs = document.querySelectorAll('#gwas-ai-tab, #gwas-public-tab');
        const gwasCorrelationSourceDb = document.getElementById('gwasCorrelationSourceDb');
        const gwasTokenGroup = document.getElementById('gwasTokenGroup');
        gwasCorrelationTabs.forEach(tab => {
            tab.addEventListener('click', () => {
                const source = tab.dataset.source;
                STATE.gwasCorrelationSource = source;
                // Update tab active state
                gwasCorrelationTabs.forEach(t => t.classList.remove('active'));
                tab.classList.add('active');
                // Update content visibility
                document.getElementById('gwas-ai-correlation').classList.toggle('active', source === 'ai');
                document.getElementById('gwas-public-correlation').classList.toggle('active', source === 'public');
            });
        });
        gwasCorrelationSourceDb.addEventListener('change', () => {
            if (gwasCorrelationSourceDb.value === 'ldlink' || gwasCorrelationSourceDb.value === '1000genomes') {
                gwasTokenGroup.style.display = 'none';
            } else {
                gwasTokenGroup.style.display = 'block';
            }
        });
    }
    function updateAIServiceInfo() {
        const service = document.getElementById('aiService').value;
        const info = document.getElementById('aiApiInfo');
        const infos = {
            deepseek: '<strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get API
key</a> (Free tier available)',
            gemini:                                    '<strong>Gemini:</strong>                                    <a
href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>',
            gpt:  '<strong>GPT:</strong>  <a  href="https://api.openai.com/v1/api-keys"  target="_blank">Get  API
key</a>'
        };
        info.innerHTML = infos[service];
    }
    function updateAICorrelationInfoHarmonization() {
```

```
            const service = document.getElementById('aiCorrelationServiceHarmonization').value;
            const info = document.getElementById('aiCorrelationApiInfoHarmonization');
            const infos = {
                deepseek: '<strong>DeepSeek:</strong> <a href="https://platform.deepseek.com/" target="_blank">Get API key</a> (Free tier available)',
                gemini: '<strong>Gemini:</strong> <a href="https://aistudio.google.com/api-keys?projectFilter=gen-lang-client-0032926246" target="_blank">Get API key</a>'
            };
            info.innerHTML = infos[service];
        }
        document.getElementById('fetchAiDataBtn').addEventListener('click', fetchAIData);
        document.getElementById('harmonizeAiBtn').addEventListener('click', () => {
            document.getElementById('aiHarmonizationConfig').style.display = 'block';
            document.getElementById('aiHarmonizationConfig').scrollIntoView({ behavior: 'smooth' });
            // Show the proceed button after config is visible
            const proceedBtn = document.createElement('button');
            proceedBtn.className = 'btn btn-primary';
            proceedBtn.style.marginTop = '15px';
            proceedBtn.textContent = 'Proceed to Harmonization';
            proceedBtn.id = 'proceedHarmonizeBtn';
            proceedBtn.addEventListener('click', harmonizeAIData);
            if (!document.getElementById('proceedHarmonizeBtn')) {
                document.getElementById('aiHarmonizationConfig').appendChild(proceedBtn);
            }
        });
        document.getElementById('copyJsonOutputBtn').addEventListener('click', () => copyToClipboard('jsonoutput'));
        document.getElementById('downloadJsonOutputBtn').addEventListener('click', () => downloadJson(document.getElementById('jsonoutput').value, 'mr_ai_generated_data.json'));
        // GWAS Solution Event Listeners
        document.getElementById('fetchGwasButton').addEventListener('click', fetchGWASData);
        document.getElementById('harmonizeGwasBtn').addEventListener('click', harmonizeGWASData);
        document.getElementById('copyJsonOutputBtn').addEventListener('click', () => copyToClipboard('jsonoutput'));
        document.getElementById('downloadJsonOutputBtn').addEventListener('click', () => downloadJson(document.getElementById('jsonoutput').value, 'mr_gwas_data.json'));
        async function fetchAIData() {
            const apiKey = document.getElementById('apiKey').value.trim();
            const aiService = document.getElementById('aiService').value;
            const exposureTraitsInput = document.getElementById('exposureTraits').value.trim();
            const exposures = exposureTraitsInput.split(',').map(t => t.trim()).filter(t => t);
            const outcome = document.getElementById('outcomeVariable').value.trim();
            const analysisType = document.querySelector('input[name="analysisType"]:checked').value;
            // Validation
            if (!apiKey) {
                alert('Please enter your API key');
                return;
```

```
        }
        if (exposures.length === 0) {
            alert('Please enter at least one exposure variable');
            return;
        }
        if (!outcome) {
            alert('Please enter an outcome variable');
            return;
        }
        if (analysisType === 'multivariate' && exposures.length < 2) {
            alert('Multivariate analysis requires at least 2 exposure variables (separate them with commas)');
            return;
        }
        // Show loading
        document.getElementById('aiLoading').style.display = 'block';
        document.getElementById('aiDataPreview').style.display = 'none';
        document.getElementById('aiHarmonizationConfig').style.display = 'none';
        try {
            const data = await callAIService(aiService, apiKey, exposures, outcome, analysisType);
            // Display AI-fetched data
            document.getElementById('aiExposureData').value = data.exposureData;
            document.getElementById('aiOutcomeData').value = data.outcomeData;
            document.getElementById('aiLoading').style.display = 'none';
            document.getElementById('aiDataPreview').style.display = 'block';
        } catch (error) {
            document.getElementById('aiLoading').style.display = 'none';
            alert(`AI data fetching failed: ${error.message}`);
        }
    }
    async function callAIService(service, apiKey, exposures, outcome, analysisType) {
        const prompt = `Fetch realistic genetic variant data for a ${analysisType} Mendelian Randomization analysis.
Exposure trait(s): ${exposures.join(', ')}
Outcome trait: ${outcome}
Fetch data in CSV format with columns: SNP,beta,se,effect_allele,other_allele,eaf
Requirements:
1. Use real SNP IDs (rs numbers)
2. Fetch 15-20 variants
3. Beta values should be realistic effect sizes (-0.5 to 0.5)
4. Standard errors should be realistic (0.01 to 0.1)
5. Use real alleles (A, T, C, G)
6. EAF should be between 0.01 and 0.99
7. Make sure SNPs match between exposure and outcome files
Return the data in this format:
EXPOSURE_DATA:
[CSV data for exposure]
```

```
OUTCOME_DATA:
[CSV data for outcome]`;
            let response;
            if (service === 'deepseek') {
                response = await fetchWithRetry('https://api.deepseek.com/chat/completions', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json',
                        'Authorization': `Bearer ${apiKey}`
                    },
                    body: JSON.stringify({
                        model: 'deepseek-chat',
                        messages: [
                            { role: 'system', content: 'You are a genetics data expert. Fetch only CSV data without markdown formatting.' },
                            { role: 'user', content: prompt }
                        ],
                        temperature: 0.7,
                        max_tokens: 4000
                    })
                });
                if (!response.ok) {
                    const error = await response.json();
                    throw new Error(error.error?.message || `API error: ${response.status}`);
                }
                const data = await response.json();
                const content = data.choices?.[0]?.message?.content;
                if (!content) {
                    throw new Error('No content received from AI');
                }
                return parseAIResponse(content);
            } else if (service === 'gemini') {
                response = await fetchWithRetry(`https://generativelanguage.googleapis.com/v1/models/gemini-1.5-flash:generateContent?key=${apiKey}`, {
                    method: 'POST',
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({
                        contents: [{ parts: [{ text: prompt }] }],
                        generationConfig: { temperature: 0.7, maxOutputTokens: 4000 }
                    })
                });
                if (!response.ok) {
                    const error = await response.json();
                    throw new Error(error.error?.message || `API error: ${response.status}`);
                }
```

```
            const data = await response.json();
            const content = data.candidates?.[0]?.content?.parts?.[0]?.text;
            if (!content) {
                throw new Error('No content received from AI');
            }
            return parseAIResponse(content);
        } else if (service === 'gpt') {
            response = await fetchWithRetry('https://api.openai.com/v1/chat/completions', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Bearer ${apiKey}`
                },
                body: JSON.stringify({
                    model: 'gpt-4o-mini',
                    messages: [
                        { role: 'system', content: 'You are a genetics data expert. Fetch only CSV data without markdown formatting.' },
                        { role: 'user', content: prompt }
                    ],
                    temperature: 0.7,
                    max_tokens: 4000
                })
            });
            if (!response.ok) {
                const error = await response.json();
                throw new Error(error.error?.message || `API error: ${response.status}`);
            }
            const data = await response.json();
            const content = data.choices?.[0]?.message?.content;
            if (!content) {
                throw new Error('No content received from AI');
            }
            return parseAIResponse(content);
        }
    }
    function parseAIResponse(content) {
        const exposureMatch = content.match(/EXPOSURE_DATA:([\s\S]*?)(?=OUTCOME_DATA:|$)/i);
        const outcomeMatch = content.match(/OUTCOME_DATA:([\s\S]*?)$/i);
        if (!exposureMatch || !outcomeMatch) {
            throw new Error('Invalid AI response format');
        }
        let exposureData = exposureMatch[1].trim();
        let outcomeData = outcomeMatch[1].trim();
        // Clean markdown
```

```
            exposureData = exposureData.replace(/```csv/gi, '').replace(/```/g, '').trim();
            outcomeData = outcomeData.replace(/```csv/gi, '').replace(/```/g, '').trim();
            return { exposureData, outcomeData };
        }
        async function harmonizeAIData() {
            const progress = document.getElementById('aiHarmonizationProgress');
            progress.style.display = 'block';
            try {
                // Step 1: Parse AI data
                updateProgressStep('ai-step-parse', 'active', 'Parsing AI-fetched data...');
                await new Promise(resolve => setTimeout(resolve, 300));
                const exposureText = document.getElementById('aiExposureData').value;
                const outcomeText = document.getElementById('aiOutcomeData').value;
                STATE.exposureData = parseCsvData(exposureText);
                STATE.outcomeData = parseCsvData(outcomeText);
                updateProgressStep('ai-step-parse', 'complete', `Parsed ${Object.keys(STATE.exposureData).length}
exposure + ${Object.keys(STATE.outcomeData).length} outcome variants`);
                // Step 2: Harmonize
                updateProgressStep('ai-step-harmonize', 'active', 'Harmonizing data...');
                await new Promise(resolve => setTimeout(resolve, 500));
                const harmonized = harmonizeData(STATE.exposureData, STATE.outcomeData);
                STATE.harmonizedData = harmonized;
                updateProgressStep('ai-step-harmonize', 'complete', `Harmonized ${harmonized.snps.length} common
SNPs`);
                // Step 3: Fetch correlation (user choice: public or AI)
                updateProgressStep('ai-step-correlation', 'active', 'Fetching correlation matrix...');
                let correlation;
                if (STATE.aiCorrelationSource === 'ai') {
                    const service = document.getElementById('aiCorrelationServiceHarmonization').value;
                    const apiKey = document.getElementById('aiCorrelationApiKeyHarmonization').value.trim();
                    if (!apiKey) {
                        throw new Error('Please provide an API key for AI correlation matrix fetching');
                    }
                    try {
                        correlation = await fetchAICorrelationMatrix(harmonized.snps, service, apiKey);
                    } catch (error) {
                        console.warn('AI correlation fetching failed, using identity matrix');
                        correlation = createIdentityMatrix(harmonized.snps);
                    }
                } else {
                    const source = document.getElementById('aiCorrelationSourceDb').value;
                    const token = document.getElementById('aiDbToken').value.trim();
                    try {
                        correlation = await fetchCorrelationMatrix(harmonized.snps, source, token);
                    } catch (error) {
```

```
                    console.warn('Public database fetching failed, using identity matrix');
                    correlation = createIdentityMatrix(harmonized.snps);
                }
            }
            STATE.correlationMatrix = correlation;
            if (correlation.source === 'identity') {
                updateProgressStep('ai-step-correlation', 'complete', 'Using identity matrix (databases unavailable)');
            } else {
                updateProgressStep('ai-step-correlation', 'complete', `Correlation matrix from ${correlation.source}`);
            }
            // Step 4: Generate JSON
            updateProgressStep('ai-step-json', 'active', 'Generating final JSON...');
            await new Promise(resolve => setTimeout(resolve, 300));
            const exposureTraitsInput = document.getElementById('exposureTraits').value.trim();
            const exposures = exposureTraitsInput.split(',').map(t => t.trim()).filter(t => t);
            const outcome = document.getElementById('outcomeVariable').value.trim();
            const finalJson = {
                snps: harmonized.snps,
                betaX: harmonized.betaX,
                betaY: harmonized.betaY,
                betaXse: harmonized.betaXse,
                betaYse: harmonized.betaYse,
                correlation: correlation.matrix,
                exposure: exposures.length === 1 ? exposures[0] : exposures,
                outcome: outcome,
                n: harmonized.snps.length,
                metadata: {
                    source: 'ai_generated',
                    ai_service: document.getElementById('aiService').value,
                    harmonized_snps: harmonized.snps.length,
                    correlation_source: correlation.source,
                    correlation_method: correlation.method,
                    timestamp: new Date().toISOString()
                }
            };
            STATE.finalJson = finalJson;
            updateProgressStep('ai-step-json', 'complete', 'JSON generated successfully');
            // Display output in global textarea
            document.getElementById('jsonoutput').value = JSON.stringify(finalJson, null, 2);
            document.getElementById('globalOutput').style.display = 'block';
        } catch (error) {
            console.error('AI harmonization error:', error);
            const activeStep = document.querySelector('#aiHarmonizationProgress .progress-step.active');
            if (activeStep) {
                updateProgressStep(activeStep.id, 'error', `Error: ${error.message}`);
```

```
            }
        }
    }
    // ============ Multi-Source GWAS Data Fetcher (from Option 4) ============
    // Trait mapping for different databases
    const traitMappings = {
        "body mass index": {
            opengwas: "ieu-a-2",
            gwas: "body mass index",
            finngen: "BMI",
            gwasatlas: "BMI",
            eqtlgen: "BMI"
        },
        "bmi": {
            opengwas: "ieu-a-2",
            gwas: "body mass index",
            finngen: "BMI",
            gwasatlas: "BMI",
            eqtlgen: "BMI"
        },
        "cholesterol": {
            opengwas: "ieu-a-300",
            gwas: "cholesterol",
            finngen: "CHOL",
            gwasatlas: "cholesterol",
            eqtlgen: "cholesterol"
        },
        "coronary artery disease": {
            opengwas: "ieu-a-7",
            gwas: "coronary artery disease",
            finngen: "I9_CHD",
            gwasatlas: "CAD",
            eqtlgen: "CAD"
        },
        "cad": {
            opengwas: "ieu-a-7",
            gwas: "coronary artery disease",
            finngen: "I9_CHD",
            gwasatlas: "CAD",
            eqtlgen: "CAD"
        },
        "type 2 diabetes": {
            opengwas: "ieu-a-26",
            gwas: "type 2 diabetes",
            finngen: "T2D",
```

```
            gwasatlas: "T2D",
            eqtlgen: "T2D"
        },
        "t2d": {
            opengwas: "ieu-a-26",
            gwas: "type 2 diabetes",
            finngen: "T2D",
            gwasatlas: "T2D",
            eqtlgen: "T2D"
        },
        "height": {
            opengwas: "ieu-a-89",
            gwas: "height",
            finngen: "HEIGHT",
            gwasatlas: "height",
            eqtlgen: "height"
        },
        "smoking": {
            opengwas: "ieu-a-22",
            gwas: "smoking behavior",
            finngen: "SMOKE",
            gwasatlas: "smoking",
            eqtlgen: "smoking"
        },
        "alzheimer": {
            opengwas: "ieu-a-297",
            gwas: "alzheimer's disease",
            finngen: "ALZHEIMER",
            gwasatlas: "AD",
            eqtlgen: "alzheimer"
        },
        "blood pressure": {
            opengwas: "ieu-a-299",
            gwas: "blood pressure",
            finngen: "BP",
            gwasatlas: "blood_pressure",
            eqtlgen: "blood_pressure"
        }
    };
    function getMappedTraitId(traitName, database) {
        const lowerTrait = traitName.toLowerCase().trim();
        // Try to find exact match in traitMappings
        if (traitMappings[lowerTrait]) {
            const mapping = traitMappings[lowerTrait];
            // Map database names to traitMappings keys
```

```
                const dbKey = database === 'ieugwas' ? 'opengwas' :
                              database === 'gwascatalog' ? 'gwas' :
                              database;
                return mapping[dbKey] || traitName;
            }
        }
        // Fuzzy matching for common terms
        for (const [key, mapping] of Object.entries(traitMappings)) {
            if (lowerTrait.includes(key) || key.includes(lowerTrait)) {
                const dbKey = database === 'ieugwas' ? 'opengwas' :
                              database === 'gwascatalog' ? 'gwas' :
                              database;
                return mapping[dbKey] || traitName;
            }
        }
        // If no mapping found, return original trait name
        console.warn(`No mapping found for trait "${traitName}" in database "${database}"`);
        return traitName;
    }
    async function fetchGWASData() {
        const exposureTraitsInput = document.getElementById('exposureTraits').value.trim();
        const exposureTraits = exposureTraitsInput.split(',').map(t => t.trim()).filter(t => t);
        const outcomeTrait = document.getElementById('outcomeVariable').value.trim();
        const pvalThreshold = parseFloat(document.getElementById('gwasPvalThreshold').value);
        // Get selected sources
        const selectedSources = Array.from(document.querySelectorAll('.checkbox-item
input[type="checkbox"]:checked'))
            .map(cb => cb.value);
        if (exposureTraits.length === 0) {
            alert('Please enter at least one exposure trait');
            return;
        }
        if (!outcomeTrait) {
            alert('Please enter an outcome trait');
            return;
        }
        if (selectedSources.length === 0) {
            alert('Please select at least one data source');
            return;
        }
        // Show progress
        document.getElementById('gwasProgressContainer').style.display = 'block';
        updateGWASProgress(0, 'Starting data fetch...');
        // Clear previous results
        document.getElementById('gwasResults').innerHTML = '';
        STATE.gwasRawData = { exposure: {}, outcome: {} };
```

```
    try {
        // Fetch exposure data
        for (let i = 0; i < exposureTraits.length; i++) {
            const trait = exposureTraits[i];
            const progress = (i / exposureTraits.length) * 50;
            updateGWASProgress(progress, `Fetching exposure data for: ${trait}`);
            const exposureData = await fetchFromSources(trait, selectedSources, pvalThreshold, 'exposure');
            STATE.gwasRawData.exposure[trait] = exposureData;
        }
        // Fetch outcome data
        updateGWASProgress(50, `Fetching outcome data for: ${outcomeTrait}`);
        const outcomeData = await fetchFromSources(outcomeTrait, selectedSources, pvalThreshold, 'outcome');
        STATE.gwasRawData.outcome[outcomeTrait] = outcomeData;
        updateGWASProgress(100, 'Data fetch complete!');
        // Display results
        displayGWASResults(exposureTraits, outcomeTrait);
        // Show harmonization section
        document.getElementById('gwasHarmonizationSection').style.display = 'block';
    } catch (error) {
        console.error('GWAS fetch error:', error);
        updateGWASProgress(0, `Error: ${error.message}`);
        alert(`Failed to fetch GWAS data: ${error.message}`);
    }
}
async function fetchFromSources(trait, sources, pvalThreshold, type) {
    const results = {
        trait: trait,
        data: {},
        sources: []
    };
    for (const source of sources) {
        try {
            await apiLimiters[source].waitForSlot();
            const data = await fetchFromSource(trait, source, pvalThreshold, type);
            if (data && data.snps && data.snps.length > 0) {
                results.data[source] = data;
                results.sources.push(source);
            }
        } catch (error) {
            console.warn(`Failed to fetch from ${source} for ${trait}:`, error);
        }
    }
    return results;
}
async function fetchFromSource(trait, source, pvalThreshold, type) {
```

```
    const cacheKey = `${source}_${trait}_${pvalThreshold}_${type}`;
    const cached = gwasCache.get(cacheKey);
    if (cached) {
        console.log(`Using cached data for ${trait} from ${source}`);
        return cached;
    }
    let data;
    switch (source) {
        case 'ieugwas':
            data = await fetchFromIEUOpenGWAS(trait, pvalThreshold);
            break;
        case 'gwascatalog':
            data = await fetchFromGWASCatalog(trait, pvalThreshold);
            break;
        case 'finngen':
            data = await fetchFromFinnGen(trait, pvalThreshold);
            break;
        case 'gwasatlas':
            data = await fetchFromGWASAtlas(trait, pvalThreshold);
            break;
        case 'eqtlgen':
            data = await fetchFromEQTLGen(trait, pvalThreshold);
            break;
        default:
            throw new Error(`Unknown source: ${source}`);
    }
    if (data) {
        gwasCache.set(cacheKey, data);
    }
    return data;
}
async function fetchFromIEUOpenGWAS(trait, pvalThreshold) {
    const mappedTrait = getMappedTraitId(trait, 'ieugwas');
    try {
        const url = `https://gwas-api.mrcieu.ac.uk/gwasinfo/?query=${encodeURIComponent(mappedTrait)}`;
        const response = await fetchWithRetry(url);
        const studies = await response.json();
        if (!studies || studies.length === 0) {
            throw new Error(`No studies found for trait: ${trait}`);
        }
        // Use the first study
        const studyId = studies[0].id;
        const summaryUrl = `https://gwas-api.mrcieu.ac.uk/associations/${studyId}`;
        const summaryResponse = await fetchWithRetry(summaryUrl);
        const summaryData = await summaryResponse.json();
```

```
            return processSummaryData(summaryData, pvalThreshold, 'ieugwas');
        } catch (error) {
            throw new Error(`IEU OpenGWAS fetch failed: ${error.message}`);
        }
    }
    async function fetchFromGWASCatalog(trait, pvalThreshold) {
        const mappedTrait = getMappedTraitId(trait, 'gwascatalog');
        try {
            const url = `https://www.ebi.ac.uk/gwas/summary-statistics/api/traits/${encodeURIComponent(mappedTrait)}/associations`;
            const response = await fetchWithRetry(url);
            const data = await response.json();
            return processSummaryData(data, pvalThreshold, 'gwascatalog');
        } catch (error) {
            throw new Error(`GWAS Catalog fetch failed: ${error.message}`);
        }
    }
    async function fetchFromFinnGen(trait, pvalThreshold) {
        const mappedTrait = getMappedTraitId(trait, 'finngen');
        try {
            // FinnGen API endpoint (example - would need actual endpoint)
            const url = `https://r8.finngen.fi/api/associations/${encodeURIComponent(mappedTrait)}`;
            const response = await fetchWithRetry(url);
            const data = await response.json();
            return processSummaryData(data, pvalThreshold, 'finngen');
        } catch (error) {
            throw new Error(`FinnGen fetch failed: ${error.message}`);
        }
    }
    async function fetchFromGWASAtlas(trait, pvalThreshold) {
        const mappedTrait = getMappedTraitId(trait, 'gwasatlas');
        try {
            // GWAS Atlas API endpoint (example)
            const url = `https://atlas.ctglab.nl/api/v1/gwas/${encodeURIComponent(mappedTrait)}`;
            const response = await fetchWithRetry(url);
            const data = await response.json();
            return processSummaryData(data, pvalThreshold, 'gwasatlas');
        } catch (error) {
            throw new Error(`GWAS Atlas fetch failed: ${error.message}`);
        }
    }
    async function fetchFromEQTLGen(trait, pvalThreshold) {
        const mappedTrait = getMappedTraitId(trait, 'eqtlgen');
        try {
            // eQTLGen API endpoint (example)
```

```
            const url = `https://eqtlgen.org/api/v1/eqtls/${encodeURIComponent(mappedTrait)}`;
            const response = await fetchWithRetry(url);
            const data = await response.json();
            return processSummaryData(data, pvalThreshold, 'eqtlgen');
        } catch (error) {
            throw new Error(`eQTLGen fetch failed: ${error.message}`);
        }
    }
    function processSummaryData(data, pvalThreshold, source) {
        if (!data || !Array.isArray(data)) {
            throw new Error('Invalid data format received from API');
        }
        const processed = {
            snps: [],
            beta: [],
            se: [],
            p: [],
            ea: [],
            nea: [],
            eaf: [],
            source: source,
            count: 0
        };
        for (const item of data) {
            if (item.p && item.p < pvalThreshold) {
                processed.snps.push(item.rsid || item.snp || item.variant);
                processed.beta.push(item.beta || item.effect_size);
                processed.se.push(item.se || item.standard_error);
                processed.p.push(item.p);
                processed.ea.push(item.ea || item.effect_allele);
                processed.nea.push(item.nea || item.other_allele);
                processed.eaf.push(item.eaf || item.allele_frequency);
                processed.count++;
            }
        }
        return processed;
    }
    function displayGWASResults(exposureTraits, outcomeTrait) {
        const resultsDiv = document.getElementById('gwasResults');
        resultsDiv.innerHTML = '';
        // Display exposure results
        exposureTraits.forEach(trait => {
            const traitData = STATE.gwasRawData.exposure[trait];
            if (!traitData) return;
            const resultItem = document.createElement('div');
```

```
                resultItem.className = 'result-item';
                let content = `<h3>Exposure: ${trait}</h3>`;
                content += `<p><strong>Sources:</strong> ${traitData.sources.join(', ')}</p>`;
                traitData.sources.forEach(source => {
                    const sourceData = traitData.data[source];
                    if (sourceData) {
                        content += `<p><strong>${source}:</strong> ${sourceData.count} significant SNPs</p>`;
                    }
                });
                // Add download buttons
                content += `<div style="margin-top: 10px;">`;
                traitData.sources.forEach(source => {
                    const sourceData = traitData.data[source];
                    if (sourceData) {
                        const csvData = convertToCSV(sourceData);
                        content += `<a href="#" class="download-button" onclick="saveDataToFile('${encodeURIComponent(csvData)}', '${trait}_${source}_exposure.csv')">Download ${source} Data</a> `;
                    }
                });
                content += `</div>`;
                // Add data preview
                content += `<div class="data-box">`;
                traitData.sources.forEach(source => {
                    const sourceData = traitData.data[source];
                    if (sourceData && sourceData.snps.length > 0) {
                        content += `<strong>${source} (first 5 SNPs):</strong>\n`;
                        for (let i = 0; i < Math.min(5, sourceData.snps.length); i++) {
                            content += `${sourceData.snps[i]}: beta=${sourceData.beta[i]}, se=${sourceData.se[i]}, p=${sourceData.p[i]}\n`;
                        }
                        content += `\n`;
                    }
                });
                content += `</div>`;
                resultItem.innerHTML = content;
                resultsDiv.appendChild(resultItem);
            });
            // Display outcome results
            const outcomeData = STATE.gwasRawData.outcome[outcomeTrait];
            if (outcomeData) {
                const resultItem = document.createElement('div');
                resultItem.className = 'result-item';
                let content = `<h3>Outcome: ${outcomeTrait}</h3>`;
                content += `<p><strong>Sources:</strong> ${outcomeData.sources.join(', ')}</p>`;
```

```
                    outcomeData.sources.forEach(source => {
                        const sourceData = outcomeData.data[source];
                        if (sourceData) {
                            content += `<p><strong>${source}:</strong> ${sourceData.count} significant SNPs</p>`;
                        }
                    });
                    // Add download buttons
                    content += `<div style="margin-top: 10px;">`;
                    outcomeData.sources.forEach(source => {
                        const sourceData = outcomeData.data[source];
                        if (sourceData) {
                            const csvData = convertToCSV(sourceData);
                            content += `<a href="#" class="download-button" onclick="saveDataToFile('${encodeURIComponent(csvData)}', '${outcomeTrait}_${source}_outcome.csv')">Download ${source} Data</a> `;
                        }
                    });
                    content += `</div>`;
                    // Add data preview
                    content += `<div class="data-box">`;
                    outcomeData.sources.forEach(source => {
                        const sourceData = outcomeData.data[source];
                        if (sourceData && sourceData.snps.length > 0) {
                            content += `<strong>${source} (first 5 SNPs):</strong>\n`;
                            for (let i = 0; i < Math.min(5, sourceData.snps.length); i++) {
                                content += `${sourceData.snps[i]}: beta=${sourceData.beta[i]}, se=${sourceData.se[i]}, p=${sourceData.p[i]}\n`;
                            }
                            content += `\n`;
                        }
                    });
                    content += `</div>`;
                    resultItem.innerHTML = content;
                    resultsDiv.appendChild(resultItem);
                }
            }
            function convertToCSV(data) {
                const headers = ['SNP', 'beta', 'se', 'p', 'effect_allele', 'other_allele', 'eaf'];
                let csv = headers.join(',') + '\n';
                for (let i = 0; i < data.snps.length; i++) {
                    const row = [
                        data.snps[i],
                        data.beta[i],
                        data.se[i],
                        data.p[i],
```

```
                    data.ea[i] || '',
                    data.nea[i] || '',
                    data.eaf[i] || ''
                ];
                csv += row.join(',') + '\n';
            }
            return csv;
        }
        function updateGWASProgress(percent, message) {
            const progressFill = document.getElementById('progressFill');
            const status = document.getElementById('gwasStatus');
            progressFill.style.width = `${percent}%`;
            progressFill.textContent = `${Math.round(percent)}%`;
            status.textContent = message;
        }
        async function harmonizeGWASData() {
            const progress = document.getElementById('gwasHarmonizationProgress');
            progress.style.display = 'block';
            try {
                // Step 1: Parse GWAS data
                updateProgressStep('gwas-step-parse', 'active', 'Parsing GWAS data...');
                await new Promise(resolve => setTimeout(resolve, 300));
                // Extract exposure and outcome data
                const exposureTraits = Object.keys(STATE.gwasRawData.exposure);
                const outcomeTrait = Object.keys(STATE.gwasRawData.outcome)[0];
                if (!exposureTraits.length || !outcomeTrait) {
                    throw new Error('No valid GWAS data found for harmonization');
                }
                // For simplicity, use the first source for each trait
                const exposureSource = STATE.gwasRawData.exposure[exposureTraits[0]].sources[0];
                const outcomeSource = STATE.gwasRawData.outcome[outcomeTrait].sources[0];
                const exposureData = STATE.gwasRawData.exposure[exposureTraits[0]].data[exposureSource];
                const outcomeData = STATE.gwasRawData.outcome[outcomeTrait].data[outcomeSource];
                // Convert to the format expected by harmonizeData
                const expData = {};
                const outData = {};
                for (let i = 0; i < exposureData.snps.length; i++) {
                    expData[exposureData.snps[i]] = {
                        rsid: exposureData.snps[i],
                        beta: exposureData.beta[i],
                        se: exposureData.se[i],
                        ea: exposureData.ea[i],
                        nea: exposureData.nea[i],
                        eaf: exposureData.eaf[i]
                    };
```

```
                }
                for (let i = 0; i < outcomeData.snps.length; i++) {
                    outData[outcomeData.snps[i]] = {
                        rsid: outcomeData.snps[i],
                        beta: outcomeData.beta[i],
                        se: outcomeData.se[i],
                        ea: outcomeData.ea[i],
                        nea: outcomeData.nea[i],
                        eaf: outcomeData.eaf[i]
                    };
                }
                STATE.exposureData = expData;
                STATE.outcomeData = outData;
                updateProgressStep('gwas-step-parse', 'complete', `Parsed ${Object.keys(expData).length} exposure + ${Object.keys(outData).length} outcome variants`);
                // Step 2: Harmonize
                updateProgressStep('gwas-step-harmonize', 'active', 'Harmonizing data...');
                await new Promise(resolve => setTimeout(resolve, 500));
                const harmonized = harmonizeData(STATE.exposureData, STATE.outcomeData);
                STATE.harmonizedData = harmonized;
                updateProgressStep('gwas-step-harmonize', 'complete', `Harmonized ${harmonized.snps.length} common SNPs`);
                // Step 3: Fetch correlation
                updateProgressStep('gwas-step-correlation', 'active', 'Fetching correlation matrix...');
                let correlation;
                if (STATE.gwasCorrelationSource === 'ai') {
                    const service = document.getElementById('gwasAiCorrelationService').value;
                    const apiKey = document.getElementById('gwasAiCorrelationApiKey').value.trim();
                    if (!apiKey) {
                        throw new Error('Please provide an API key for AI correlation matrix fetching');
                    }
                    try {
                        correlation = await fetchAICorrelationMatrix(harmonized.snps, service, apiKey);
                    } catch (error) {
                        console.warn('AI correlation fetching failed, using identity matrix');
                        correlation = createIdentityMatrix(harmonized.snps);
                    }
                } else {
                    const source = document.getElementById('gwasCorrelationSourceDb').value;
                    const token = document.getElementById('gwasDbToken').value.trim();
                    try {
                        correlation = await fetchCorrelationMatrix(harmonized.snps, source, token);
                    } catch (error) {
                        console.warn('Public database fetching failed, using identity matrix');
                        correlation = createIdentityMatrix(harmonized.snps);
```

```
                    }
                }
                STATE.correlationMatrix = correlation;
                if (correlation.source === 'identity') {
                    updateProgressStep('gwas-step-correlation', 'complete', 'Using identity matrix (databases unavailable)');
                } else {
                    updateProgressStep('gwas-step-correlation', 'complete', `Correlation matrix from ${correlation.source}`);
                }
                // Step 4: Generate JSON
                updateProgressStep('gwas-step-json', 'active', 'Generating final JSON...');
                await new Promise(resolve => setTimeout(resolve, 300));
                const finalJson = {
                    snps: harmonized.snps,
                    betaX: harmonized.betaX,
                    betaY: harmonized.betaY,
                    betaXse: harmonized.betaXse,
                    betaYse: harmonized.betaYse,
                    correlation: correlation.matrix,
                    exposure: exposureTraits.length === 1 ? exposureTraits[0] : exposureTraits,
                    outcome: outcomeTrait,
                    n: harmonized.snps.length,
                    metadata: {
                        source: 'gwas_fetched',
                        exposure_sources: exposureTraits.map(t => STATE.gwasRawData.exposure[t].sources),
                        outcome_sources: STATE.gwasRawData.outcome[outcomeTrait].sources,
                        harmonized_snps: harmonized.snps.length,
                        correlation_source: correlation.source,
                        correlation_method: correlation.method,
                        timestamp: new Date().toISOString()
                    }
                };
                STATE.finalJson = finalJson;
                updateProgressStep('gwas-step-json', 'complete', 'JSON generated successfully');
                // Display output in global textarea
                document.getElementById('jsonoutput').value = JSON.stringify(finalJson, null, 2);
                document.getElementById('globalOutput').style.display = 'block';
            } catch (error) {
                console.error('GWAS harmonization error:', error);
                const activeStep = document.querySelector('#gwasHarmonizationProgress .progress-step.active');
                if (activeStep) {
                    updateProgressStep(activeStep.id, 'error', `Error: ${error.message}`);
                }
            }
        }
```

```
// ============ Utility Functions ============
function copyToClipboard(textareaId) {
    const textarea = document.getElementById(textareaId);
    textarea.select();
    textarea.setSelectionRange(0, 99999); // For mobile devices
    try {
        document.execCommand('copy');
        // Show feedback
        const btn = event.target;
        const originalText = btn.textContent;
        btn.textContent = 'Copied!';
        btn.style.background = '#28a745';
        setTimeout(() => {
            btn.textContent = originalText;
            btn.style.background = '';
        }, 2000);
        // Show success message
        const status = document.createElement('div');
        status.className = 'status success';
        status.textContent = 'JSON copied to clipboard!';
        status.style.marginTop = '10px';
        btn.parentNode.insertBefore(status, btn.nextSibling);
        setTimeout(() => status.remove(), 3000);
    } catch (err) {
        console.error('Failed to copy text: ', err);
        alert('Failed to copy to clipboard. Please select and copy manually.');
    }
}
function downloadJson(jsonString, filename) {
    try {
        // Validate JSON before download
        const parsed = JSON.parse(jsonString);
        const blob = new Blob([JSON.stringify(parsed, null, 2)], { type: 'application/json' });
        const url = URL.createObjectURL(blob);
        const a = document.createElement('a');
        a.href = url;
        a.download = filename;
        document.body.appendChild(a);
        a.click();
        document.body.removeChild(a);
        URL.revokeObjectURL(url);
        // Show success feedback
        const btn = event.target;
        const originalText = btn.textContent;
        btn.textContent = 'Downloaded!';
```

```
                btn.style.background = '#28a745';
                setTimeout(() => {
                    btn.textContent = originalText;
                    btn.style.background = '';
                }, 2000);
            } catch (error) {
                console.error('JSON download error:', error);
                alert('Invalid JSON format. Please validate before downloading.');
            }
        }
        function saveDataToFile(data, filename) {
            const decodedData = decodeURIComponent(data);
            const blob = new Blob([decodedData], { type: 'text/csv;charset=utf-8;' });
            const url = URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
            a.download = filename;
            document.body.appendChild(a);
            a.click();
            document.body.removeChild(a);
            URL.revokeObjectURL(url);
        }
        // Global error handler for uncaught errors
        window.addEventListener('error', function(e) {
            console.error('Global error:', e.error);
            // Show user-friendly error message
            const errorDiv = document.createElement('div');
            errorDiv.style.cssText = `
                position: fixed; top: 20px; right: 20px; z-index: 10000;
                background: #dc3545; color: white; padding: 15px; border-radius: 8px;
                box-shadow: 0 4px 12px rgba(0,0,0,0.3); max-width: 400px;
                font-family: inherit; font-size: 14px;
            `;
            errorDiv.innerHTML = `
                <strong>An unexpected error occurred</strong><br>
                <small>${e.error.message || 'Unknown error'}</small><br><br>
                <button onclick="this.parentElement.remove()" style="background: none; border: 1px solid white; color: white; padding: 5px 10px; border-radius: 4px; cursor: pointer;">Dismiss</button>
            `;
            document.body.appendChild(errorDiv);
            // Auto-dismiss after 10 seconds
            setTimeout(() => {
                if (errorDiv.parentNode) {
                    errorDiv.remove();
                }
```

```
            }, 10000);
        });
        // Initialize tooltips and help text
        document.addEventListener('DOMContentLoaded', function() {
            // Add tooltips for complex fields
            const helpElements = document.querySelectorAll('.help-text');
            helpElements.forEach(el => {
                el.title = el.textContent; // Make help text available as tooltip
            });
            console.log('MR Data Fetcher Enhanced v2.0 loaded successfully');
            console.log('Features: Advanced validation, caching, rate limiting, retry logic, CORS handling');
        });
    </script>
</body>
</html>
```

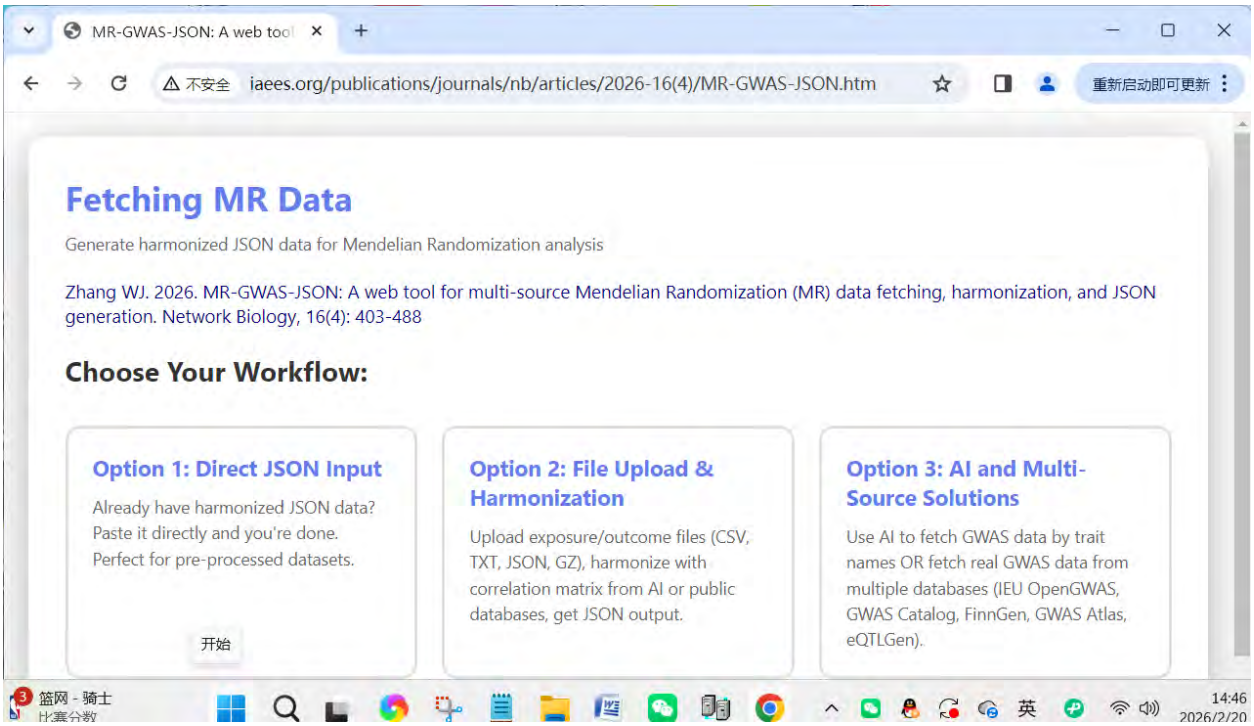

Fig. 1 MR-GWAS-JSON data fetcher.

6 Demo and Explanation

6.1 AI models

This tool uses DeepSeek and Gemini as the AI models. Here I take DeepSeek as the example. Register in https://platform.deepseek.com to get the API key and pay for API use (a very little cost, e.g., 5 cents).

6.2 Data fetching

Use DeepSeek to fetch MR data. Exposure: smoking; Outcome: lung cancer. The fetched exposure and out come data are as follows

Exposure data:
SNP,beta,se,effect_allele,other_allele,eaf
rs1051730,0.324,0.023,T,C,0.362
rs16969968,0.287,0.024,G,A,0.364
rs8034191,0.215,0.021,C,T,0.281
rs6495309,0.183,0.022,T,C,0.417
rs1329650,0.198,0.023,G,C,0.301
rs3733829,0.176,0.022,G,A,0.455
rs7937,0.142,0.021,A,G,0.388
rs4105144,0.155,0.022,C,T,0.332
rs6474412,0.121,0.021,T,C,0.294
rs1317286,0.134,0.022,A,G,0.376
rs2656052,0.108,0.021,G,A,0.408
rs667282,0.095,0.022,C,T,0.267
rs13280604,0.088,0.021,A,G,0.319
rs12440014,0.077,0.022,T,C,0.352
rs7260329,0.069,0.021,C,T,0.241
rs8040868,0.062,0.022,G,A,0.285
rs11636753,0.055,0.021,T,C,0.193
rs11064437,0.048,0.022,A,G,0.224

Outcome data:
SNP,beta,se,effect_allele,other_allele,eaf
rs1051730,0.324,0.023,T,C,0.362
rs16969968,0.287,0.024,G,A,0.364
rs8034191,0.215,0.021,C,T,0.281
rs6495309,0.183,0.022,T,C,0.417
rs1329650,0.198,0.023,G,C,0.301
rs3733829,0.176,0.022,G,A,0.455
rs7937,0.142,0.021,A,G,0.388
rs4105144,0.155,0.022,C,T,0.332
rs6474412,0.121,0.021,T,C,0.294
rs1317286,0.134,0.022,A,G,0.376
rs2656052,0.108,0.021,G,A,0.408
rs667282,0.095,0.022,C,T,0.267
rs13280604,0.088,0.021,A,G,0.319
rs12440014,0.077,0.022,T,C,0.352
rs7260329,0.069,0.021,C,T,0.241
rs8040868,0.062,0.022,G,A,0.285
rs11636753,0.055,0.021,T,C,0.193
rs11064437,0.048,0.022,A,G,0.224

After harmonization with AI, the JSON data for MR analysis are achieved as follows:

```
{
  "snps": [
    "rs1051730",
    "rs16969968",
    "rs8034191",
    "rs6495309",
    "rs1329650",
    "rs3733829",
    "rs7937",
    "rs4105144",
    "rs6474412",
    "rs1317286",
    "rs2656052",
    "rs667282",
    "rs13280604",
    "rs12440014",
    "rs7260329",
    "rs8040868",
    "rs11636753",
    "rs11064437"
  ],
  "betaX": [
    0.324,
    0.287,
    0.215,
    0.183,
    0.198,
    0.176,
    0.142,
    0.155,
    0.121,
    0.134,
    0.108,
    0.095,
    0.088,
    0.077,
    0.069,
    0.062,
    0.055,
    0.048
  ],
  "betaY": [
    0.324,
```

0.287,
0.215,
0.183,
0.198,
0.176,
0.142,
0.155,
0.121,
0.134,
0.108,
0.095,
0.088,
0.077,
0.069,
0.062,
0.055,
0.048
],
"betaXse": [
0.023,
0.024,
0.021,
0.022,
0.023,
0.022,
0.021,
0.022,
0.021,
0.022,
0.021,
0.022,
0.021,
0.022,
0.021,
0.022,
0.021,
0.022
],
"betaYse": [
0.023,
0.024,
0.021,
0.022,
0.023,

```
      0.022,
      0.021,
      0.022,
      0.021,
      0.022,
      0.021,
      0.022,
      0.021,
      0.022,
      0.021,
      0.022,
      0.021,
      0.022
    ],
    "correlation": [
      [
        1,
        0.92,
        0.85,
        0.03,
        0.02,
        0.01,
        0.04,
        0.02,
        0.01,
        0.01,
        0.02,
        0.01,
        0.01,
        0.01,
        0.02,
        0.01,
        0.01,
        0.01
      ],
      [
        0.92,
        1,
        0.88,
        0.02,
        0.01,
        0.01,
        0.03,
        0.02,
```

0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01
],
[
0.85,
0.88,
1,
0.02,
0.01,
0.01,
0.03,
0.02,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01
],
[
0.03,
0.02,
0.02,
1,
0.78,
0.65,
0.04,
0.03,
0.02,
0.02,
0.03,
0.02,

0.02,
0.02,
0.03,
0.02,
0.02,
0.02
],
[
0.02,
0.01,
0.01,
0.78,
1,
0.82,
0.03,
0.02,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01
],
[
0.01,
0.01,
0.01,
0.65,
0.82,
1,
0.02,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,
0.01,

0.01,
0.01
],
[
0.04,
0.03,
0.03,
0.04,
0.03,
0.02,
1,
0.85,
0.72,
0.68,
0.05,
0.03,
0.02,
0.02,
0.04,
0.03,
0.02,
0.02
],
[
0.02,
0.02,
0.02,
0.03,
0.02,
0.01,
0.85,
1,
0.78,
0.74,
0.04,
0.02,
0.01,
0.01,
0.03,
0.02,
0.01,
0.01
],
[

0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.72,
0.78,
1,
0.86,
0.03,
0.02,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.68,
0.74,
0.86,
1,
0.03,
0.02,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01
],
[
0.02,
0.01,
0.01,
0.03,

0.02,
0.01,
0.05,
0.04,
0.03,
0.03,
1,
0.75,
0.62,
0.58,
0.06,
0.04,
0.03,
0.03
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.03,
0.02,
0.02,
0.02,
0.75,
1,
0.68,
0.64,
0.04,
0.03,
0.02,
0.02
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.02,
0.01,

0.01,
0.01,
0.62,
0.68,
1,
0.82,
0.03,
0.02,
0.01,
0.01
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01,
0.58,
0.64,
0.82,
1,
0.03,
0.02,
0.01,
0.01
],
[
0.02,
0.01,
0.01,
0.03,
0.02,
0.01,
0.04,
0.03,
0.02,
0.02,
0.06,
0.04,

0.03,
0.03,
1,
0.78,
0.65,
0.61
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.03,
0.02,
0.01,
0.01,
0.04,
0.03,
0.02,
0.02,
0.78,
1,
0.72,
0.68
],
[
0.01,
0.01,
0.01,
0.02,
0.01,
0.01,
0.02,
0.01,
0.01,
0.01,
0.03,
0.02,
0.01,
0.01,
0.65,
0.72,

```
        1,
        0.84
      ],
      [
        0.01,
        0.01,
        0.01,
        0.02,
        0.01,
        0.01,
        0.02,
        0.01,
        0.01,
        0.01,
        0.03,
        0.02,
        0.01,
        0.01,
        0.61,
        0.68,
        0.84,
        1
      ]
    ],
    "exposure": "smoking",
    "outcome": "lung cancer",
    "n": 18,
    "metadata": {
      "source": "ai_generated",
      "ai_service": "gpt",
      "harmonized_snps": 18,
      "correlation_source": "ai_deepseek",
      "correlation_method": "ld_based_ai_estimation",
      "timestamp": "2026-02-19T02:37:55.554Z"
    }
}
```


7 Discussion

The current HTML-based MR data fetcher is a robust, client-side application with strong features like multi-workflow support (JSON input, file upload, AI/multi-source GWAS fetching), rate limiting, caching, and retry logic. However, several enhancements could improve usability, reliability, and scalability.

Backend Integration for CORS and APIs: Shift sensitive operations (e.g., GWAS API calls, AI requests) to a Node.js/Flask backend to bypass browser CORS restrictions and handle authentication securely. This

would enable real-time data processing without exposing API keys client-side, reducing security risks and improving performance for large datasets.

Enhanced Data Validation and Visualization: Add interactive previews with libraries like DataTables or Plotly for GWAS results, including scatter plots of beta values and p-value histograms. Implement stricter validation (e.g., allele frequency checks, beta/SE bounds) with user-editable fixes, and support for palindromic SNP handling in harmonization.

Advanced AI Features: Integrate more AI models (e.g., via Hugging Face) for automated trait mapping and synthetic data generation. Add natural language querying (e.g., "fetch BMI and diabetes data") using NLP, and include uncertainty estimation for AI-generated correlations.

UI/UX and Accessibility: Adopt a modern framework like React/Vue for dynamic components, reducing code bloat. Add dark mode, responsive mobile layouts, progress indicators with estimated times, and ARIA labels for screen readers. Include a tutorial modal and export options (e.g., R/Python scripts for MR analysis).

Performance and Extensibility: Optimize for large files with Web Workers for parsing/harmonization. Add plugin support for new databases (e.g., UK Biobank) and export to standard formats like MR-Base inputs. Implement unit tests and error logging to a service like Sentry.

These changes would make the tool more production-ready, user-friendly, and extensible, potentially increasing adoption in genomic research.

References

Bowden J, Davey Smith G, Burgess S. 2015. Mendelian randomization with invalid instruments: effect estimation and bias detection through Egger regression. International Journal of Epidemiology, 44(2): 512-525. doi: https://doi.org/10.1093/ije/dyv080

Burgess S, Bowden J. 2015. Integrating summarized data from multiple genetic variants in Mendelian randomization: bias and coverage properties of inverse-variance weighted methods. arXiv, 1512.04486

Burgess S, Bowden J, Dudbridge F, Thompson SG. 2016. Robust instrumental variable methods using multiple candidate instruments with application to Mendelian randomization. arXiv, 1606.03729

Burgess S, Butterworth AS, Thompson SG. 2013. Mendelian randomization analysis with multiple genetic variants using summarized data. Genetic Epidemiology, 37: 658-665. doi: https://doi.org/10.1002/gepi.21758

Davies NM, Holmes MV, Davey Smith G. 2018. Reading Mendelian randomisation studies: a guide, glossary, and checklist for clinicians. BMJ, 362: k601. doi: https://doi.org/10.1136/bmj.k601

Extance A. 2025. AI-generated medical data can sidestep usual ethics review, universities say. Nature News, 10 Sept 2025. doi: https://doi.org/10.1038/d41586-025-02911-1

GWASLab. 2024. Mendelian Randomization Series No. 1: Basic Concepts Mendelian randomization. https://gwaslab.org/2021/06/24/mr/

Hartwig FP, Smith GD, Bowden J. 2017. Robust inference in summary data Mendelian randomization via the zero modal pleiotropy assumption. International Journal of Epidemiology, 46(6): 1985-1998. doi: https://doi.org/10.1093/ije/dyx102

Hemani G, Bowden J, Davey Smith G. 2018. Evaluating the potential role of pleiotropy in Mendelian randomization studies. Human Molecular Genetics, 27(R2): R195-R208. doi: https://doi.org/10.1093/hmg/ddy163

Rajkomar A, Dean J, Kohane I. 2019. Machine learning in medicine. New England Journal of Medicine, 380(14): 1347-1358. doi: https://doi.org/10.1056/NEJMra1814259

Tam V, Patel N, Turcotte M, et al. 2019. Benefits and limitations of genome-wide association studies. Nature Reviews Genetics, 20(8): 467-484. doi: https://doi.org/10.1038/s41576-019-0127-1

Topol EJ. 2019. High-performance medicine: the convergence of human and artificial intelligence. Nature Medicine, 25(1): 44-56. doi: https://doi.org/10.1038/s41591-018-0300-7

Visscher PM, Brown MA, McCarthy MI, Yang J. 2012. Five years of GWAS discovery. American Journal of Human Genetics, 90(1): 7-24. doi: https://doi.org/10.1016/j.ajhg.2011.11.029

Zhang WJ. 2025a. A web-based data generator for Mendelian Randomization (MR) analysis. Network Pharmacology, 10(3-4): 14-65. http://www.iaees.org/publications/journals/np/articles/2025-10(3-4)/1-Zhang-Abstract.asp

Zhang WJ. 2025b. Mendelian randomization: Principles and methods. Network Biology, 15(2): 24-47. http://www.iaees.org/publications/journals/nb/articles/2025-15(2)/1-Zhang-Abstract.asp

Zhang WJ. 2026a. A GWAS data fetcher with AI for Mendelian Randomization analysis. Network Pharmacology, 11(3-4): 62-89. http://www.iaees.org/publications/journals/np/articles/2026-11(3-4)/1-Zhang-Abstract.asp

Zhang WJ. 2026b. AI-driven assessment of animal adaptation to climate change: The web tool based on physiological, morphological, behavioral, and genetic indicators of animal species. Computational Ecology and Software, 16(1): 58-98. http://www.iaees.org/publications/journals/ces/articles/2026-16(1)/3-Zhang-Abstract.asp

Zhang WJ. 2026c. AI-driven assessment of plant adaptation to climate change: The web tool based on physiological, biological, morphological, and genetic indicators of plant species. Computational Ecology and Software, 16(2): 99-153. http://www.iaees.org/publications/journals/ces/articles/2026-16(2)/1-Zhang-Abstract.asp

Zhang WJ. 2026d. DR SUN YATSEN: An AI-Based medical clinic. Network Biology, 16(3): 117-267. http://www.iaees.org/publications/journals/nb/articles/2026-16(3)/2-Zhang-Abstract.asp

Zhang WJ. 2026e. MR PHARMACIST: AI-Based medication consultation platform. Network Biology, 16(3): 268-388. http://www.iaees.org/publications/journals/nb/articles/2026-16(3)/3-Zhang-Abstract.asp

Zhang WJ. 2026f. Species Distribution Finder: The AI-driven web tool to discover where species live around the world. Computational Ecology and Software, 16(2): 172-197. http://www.iaees.org/publications/journals/ces/articles/2026-16(2)/3-Zhang-Abstract.asp

Zhang WJ. 2026g. TraitGenePathAna: The AI-Powered biological trait analysis platform. Network Biology, 16(2): 49-81. http://www.iaees.org/publications/journals/nb/articles/2026-16(2)/2-Zhang-Abstract.asp

Zhang WJ, Qi YH. 2026. Fetching GWAS (Genome-Wide Association Study) data via AI: A web tool to synthesize genotype, phenotype, and summary statistics. Ornamental and Medicinal Plants, 9(1-4): 1-21. http://www.iaees.org/publications/journals/omp/articles/2026-9(1-4)/1-Zhang-Abstract.asp