

Article

Fetching GWAS (Genome-Wide Association Study) data via AI: A web tool to synthesize genotype, phenotype, and summary statistics

WenJun Zhang¹, Yanhong Qi²

¹School of Life Sciences, Sun Yat-sen University, Guangzhou 510275, China; International Academy of Ecology and Environmental Sciences, Hong Kong

²Libraries of Sun Yat-sen University, Sun Yat-sen University, Guangzhou, China

E-mail: zhwj@mail.sysu.edu.cn, wjzhang@iaees.org

Received 1 November 2025; Accepted 16 November 2025; Published online 1 December 2025; Published 1 September 2026



Abstract

A GWAS (Genome-Wide Association Study) data fetcher via AI was developed in present study. It is a web tool that generates realistic synthetic GWAS data based on user inputs via AI APIs (DeepSeek, Google Gemini, or OpenAI GPT). It outputs three data components: (1) summary statistics: an array of SNP records (CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO), (2) phenotype data: an array of individual records (FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH), and (3) metadata: a descriptive string containing genotyping platform, QC protocols, genome build, and population notes. By leveraging AI to generate realistic data, users can practice analysis without accessing restricted genetic databases, test analysis pipelines safely, learn GWAS data structure and format, and develop and validate bioinformatics tools.

Keywords GWAS (Genome-Wide Association Study); data fetcher; Artificial Intelligence (AI); Large Language Model (LLM); JavaScript/HTML.

Ornamental and Medicinal Plants

ISSN 2522-3682

URL: <http://www.iaees.org/publications/journals/omp/online-version.asp>

RSS: <http://www.iaees.org/publications/journals/omp/rss.xml>

E-mail: omp@iaees.org

Editor-in-Chief: WenJun Zhang

Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction

The Genome-Wide Association Study (GWAS) data reflect the effects of SNPs on phenotypes (Zhang, 2025a-b, 2026a-b). GWAS data provide information on the association between genotype and phenotype. GWAS data can be used to identify SNPs that are significantly associated with specific phenotypes (GWASLab, 2024). In the earlier study, I developed a web tool for fetching GWAS data from multiple public databases (Zhang, 2026b). In principle, it is a most reliable way to obtain GWAS data. However accessing public databases will occasionally meet strict restrictions. Due to some significant advantages, more and more researchers are expected to use the data fetched and generated by AI (Extance, 2025). Latterly, I developed a GWAS (Genome-Wide Association Study) data fetcher with AI in another study (Zhang, 2026a). It is a web-based tool that generates genetic variant data using AI. It supports several AI services as DeepSeek,

Google Gemini, and OpenAI GPT, etc.

In present study, a GWAS (Genome-Wide Association Study) data fetcher via AI was developed. It is a web tool that generates realistic synthetic GWAS data based on user inputs via AI APIs (DeepSeek, Google Gemini, or OpenAI GPT). It will output three data components: (1) summary statistics: an array of SNP records, (2) phenotype data: an array of 10 individual records, and (3) metadata: a descriptive string containing genotyping platform, QC protocols, genome build, and population notes. It is a useful tool to generate realistic data for practice analysis without accessing restricted public databases,

2 Algorithmic Description

2.1 Goal

- Generate realistic synthetic GWAS data based on user inputs via AI APIs (OpenAI, DeepSeek, or Google Gemini).
- Output three data components:
 - summary_stats: an array of SNP records (CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO)
 - phenotype_data: an array of individual records (FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH)
 - metadata: a descriptive string containing genotyping platform, QC protocols, genome build, and population notes
- Render results in the UI and provide downloads.

2.2 Core components and flow

A. Configuration and state

- API_URLS: mapping of AI providers to their API endpoints:
 - deepseek: DeepSeek API
 - openai: OpenAI chat/completions
 - gemini: Google Gemini generateContent endpoint
- generatedData: in-page cache for downloading and rendering
 - summaryStats: array of SNP objects
 - phenotypes: array of phenotype objects
 - metadata: string
- Event handling: form submission triggers the generation workflow; tab switching controls result views.

B. Input collection and system/user prompts

- Inputs collected:
 - phenotype
 - ancestry
 - snpId
 - region
 - gene
 - pval
 - maf
 - info
- System prompt (systemPrompt): instructs the model to produce REALISTIC SYNTHETIC GWAS DATA and return a valid JSON object with keys:

- summary_stats: array of objects with fields CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO
- phenotype_data: array of individuals with fields FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH
- metadata: string containing Genotyping Platform, QC Protocols, Genome Build, population structure notes
- User prompt (userPrompt): describes the generated data context using inputs (trait, ancestry, SNP/gene/region, and filters). It enforces strict JSON formatting.

C. API invocation logic

- If provider is gemini, call callGemini; otherwise use callOpenAICompatible for OpenAI/DeepSeek-like providers.
- Request structure:
 - Uses a chat-style API with system and user messages
 - temperature = 0.5, max_tokens = 2500 (adjustable)
- Response handling:
 - Extract content from the AI response, attempt to parse as JSON via cleanJsonString
 - If parsing fails, throw an error with guidance to inspect raw output

D. Data parsing and cleaning

- cleanJsonString(str): remove markdown blocks (json,), trim, then JSON.parse. If parsing fails, throw a descriptive error.
- Extracted data maps to:
 - data.summary_stats -> generatedData.summaryStats
 - data.phenotype_data -> generatedData.phenotypes
 - data.metadata -> generatedData.metadata

E. Rendering

- Render three data views:
 - Summary Statistics table (#sumStatsTable): CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO
 - Phenotype Data table (#pheTable): FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH
 - Metadata display (#metadataContent)
- Result area shown after successful render
- Highlight P-values: values below 5e-8 displayed in red and bold for emphasis

F. Download functionality

- Supports:
 - sumstats: gwas_sumstats.tsv
 - phe: phenotypes.phe
- Implementation:
 - Convert data to TSV format with headers
 - Create a Blob, generate a temporary URL, and trigger download

G. Error handling and UX

- Validate API Key presence; alert if missing
- Show a loader during API calls; disable the fetch button to prevent multiple submissions
- Catch and alert errors with informative messages

H. Data structure (generated data)

- summary_stats: array of objects with:
 - CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO
- phenotype_data: array of objects with:
 - FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH
- metadata: string

3 User Guide

3.1 Goal

- Use the web app to generate and download synthetic GWAS data for demonstration, teaching, or offline analysis.

Prerequisites

- A valid API key for your chosen AI provider (OpenAI, DeepSeek, or Google Gemini).

3.2 Steps

1. Open the page and choose an AI provider
 - In the top-left “AI Provider” dropdown, select:
 - DeepSeek API
 - OpenAI GPT-4/3.5
 - Google Gemini
2. Enter your API Key
 - Paste your provider's key into the “API Key” field (e.g., a sk- key for OpenAI).
3. Configure GWAS generation parameters
 - Phenotype / Trait: enter the trait of interest (e.g., Type 2 Diabetes, Height)
 - Population / Ancestry: select EUR, EAS, SAS, AFR, AMR, or Multi-ancestry
 - Variant ID (Optional): e.g., rs7903146; otherwise leave as default
 - Target Gene (Optional): e.g., TCF7L2; otherwise leave as default
 - Genomic Region: e.g., 10:114500000-115000000; otherwise leave blank
 - P-value Threshold: choose one
 - Genome-wide sig (< 5e-8)
 - Suggestive (< 1e-5)
 - Nominal (< 0.05)
 - Min. Allele Frequency (MAF): default 0.01
 - Min. Info Score: default 0.8
4. Generate data
 - Click the “Fetch Data” button.
 - The system sends prompts to the chosen AI provider and returns a JSON object with synthetic data.
5. View and download results
 - After generation, three tabs appear:
 - Summary Statistics: view and download as a TSV file (Download .TSV)
 - Phenotype File: view and download as a PHE file (Download .PHE)
 - Metadata & Logs: view the metadata string
 - Downloads
 - sumstats: gwas_sumstats.tsv
 - phe: phenotypes.phe
 - Files use tab-separated values (TSV) formatting for easy import into GWAS tools or spreadsheets.

6. Interpreting the results
 - Summary Statistics table includes:
 - CHR, SNP, BP, A1, A2, FRQ_A1, BETA, SE, P, N, INFO
 - P-values less than 5e-8 are highlighted in red to indicate significance
 - Phenotype Data table includes:
 - FID, IID, PHE, SEX, AGE, PC1, PC2, BATCH
 - PHE can be binary (0/1) or continuous, depending on trait
 - Metadata provides contextual details about the simulated data generation process

3.3 Notes and tips

- The tool relies on AI to generate synthetic data; use the prompts to guide realism.
- If parsing errors occur, ensure your API response is valid JSON and that the content does not include extraneous text; the app attempts to clean common markdown blocks.
- You can customize thresholds (pval, maf, info) for more conservative or broader synthetic datasets.
- The app emphasizes reproducibility by using a fixed temperature and a reasonable token limit.

4 Codes

The following are the HTML+JavaScript codes of the GWAS data fetcher via AI ([http://www.iaees.org/publications/journals/omp/articles/2026-9\(1-4\)/GWASdataFetchAI.htm](http://www.iaees.org/publications/journals/omp/articles/2026-9(1-4)/GWASdataFetchAI.htm); Fig. 1):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>AI-Powered GWAS Data Fetcher</title>
  <style>
    :root {
      --primary: #2563eb;
      --secondary: #1e40af;
      --bg: #f3f4f6;
      --card: #ffffff;
      --text: #1f2937;
      --border: #e5e7eb;
    }

    body {
      font-family: 'Segoe UI', Roboto, Helvetica, Arial, sans-serif;
      background-color: var(--bg);
      color: var(--text);
      margin: 0;
      padding: 20px;
      line-height: 1.5;
    }

    .container {
      max-width: 1200px;
      margin: 0 auto;
    }
  </style>
</head>
```

```
}

/* Header */
header {
  background-color: var(--primary);
  color: white;
  padding: 20px;
  border-radius: 8px;
  margin-bottom: 20px;
  box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);
}

h1 { margin: 0; font-size: 1.5rem; }
p.subtitle { margin: 5px 0 0; opacity: 0.9; font-size: 0.9rem; }

/* API Configuration */
.config-panel {
  background: var(--card);
  padding: 20px;
  border-radius: 8px;
  border: 1px solid var(--border);
  margin-bottom: 20px;
  display: flex;
  gap: 15px;
  flex-wrap: wrap;
  align-items: flex-end;
}

.form-group {
  display: flex;
  flex-direction: column;
  gap: 5px;
  flex: 1;
  min-width: 200px;
}

label {
  font-weight: 600;
  font-size: 0.85rem;
  color: #4b5563;
}

input, select {
  padding: 10px;
  border: 1px solid var(--border);
  border-radius: 6px;
  font-size: 1rem;
}

input:focus, select:focus {
```

```
    outline: 2px solid var(--primary);
    border-color: transparent;
}

/* Main Input Grid */
.main-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
    gap: 20px;
    margin-bottom: 20px;
}

.card {
    background: var(--card);
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

.card h3 {
    margin-top: 0;
    border-bottom: 2px solid var(--bg);
    padding-bottom: 10px;
    color: var(--primary);
}

/* Buttons */
.btn {
    background-color: var(--primary);
    color: white;
    padding: 12px 24px;
    border: none;
    border-radius: 6px;
    cursor: pointer;
    font-weight: 600;
    transition: background 0.2s;
    width: 100%;
}

.btn:hover { background-color: var(--secondary); }
.btn:disabled { background-color: #9ca3af; cursor: not-allowed; }

.btn-download {
    background-color: #059669;
    margin-top: 10px;
    width: auto;
    font-size: 0.85rem;
    padding: 8px 16px;
}
```

```
/* Results Area */
#resultsArea {
    display: none;
}

.tabs {
    display: flex;
    gap: 10px;
    margin-bottom: 15px;
}

.tab-btn {
    padding: 10px 20px;
    background: #d1d5db;
    border: none;
    border-radius: 6px;
    cursor: pointer;
}

.tab-btn.active {
    background: var(--primary);
    color: white;
}

.tab-content {
    display: none;
    background: var(--card);
    padding: 20px;
    border-radius: 8px;
    overflow-x: auto;
}

.tab-content.active { display: block; }

/* Tables */
table {
    width: 100%;
    border-collapse: collapse;
    font-size: 0.9rem;
    min-width: 800px;
}

th, td {
    padding: 10px;
    border-bottom: 1px solid var(--border);
    text-align: left;
    font-family: 'Courier New', monospace;
}

th {
```

```

        background-color: #f9fafb;
        font-weight: 700;
    }

    /* Loading Spinner */
    .loader {
        border: 4px solid #f3f3f3;
        border-top: 4px solid var(--primary);
        border-radius: 50%;
        width: 30px;
        height: 30px;
        animation: spin 1s linear infinite;
        margin: 20px auto;
        display: none;
    }

    @keyframes spin { 0% { transform: rotate(0deg); } 100% { transform: rotate(360deg); } }

    .metadata-box {
        background: #f8fafc;
        padding: 15px;
        border: 1px solid var(--border);
        border-radius: 6px;
        font-family: monospace;
        white-space: pre-wrap;
    }
</style>
</head>
<body>

<div class="container">
    <header>
        <h1> AI GWAS Data Fetcher</h1>
        <p class="subtitle">Synthesize Genotype, Phenotype, and Summary Statistics via AI APIs</p>
    </header>
    <a href="http://www.iaees.org/publications/journals/omp/articles/2026-9(1-4)/1-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</a>
    <!-- API Configuration -->
    <div class="config-panel">
        <div class="form-group">
            <label for="apiProvider">AI Provider</label>
            <select id="apiProvider">
                <option value="deepseek">DeepSeek API</option>
                <option value="openai">OpenAI GPT-4/3.5</option>
                <option value="gemini">Google Gemini</option>
            </select>
        </div>
        <div class="form-group" style="flex: 2;">
            <label for="apiKey">API Key</label>

```

```

    <input type="password" id="apiKey" placeholder="">
  </div>
</div>
<div>
  □ How to get AI API keys:<br>
  • DeepSeek: <a href="https://platform.deepseek.com/">platform.deepseek.com</a><br>
  • Gemini: <a href="https://makersuite.google.com/app/apikey">makersuite.google.com/app/apikey</a><br>
  • GPT: <a href="https://platform.openai.com/api-keys">platform.openai.com/api-keys</a>
</div>

<form id="gwasForm">
  <div class="main-grid">
    <!-- 1. Core Inquiry -->
    <div class="card">
      <h3>1. Core Inquiry Variables</h3>
      <div class="form-group">
        <label for="phenotype">Phenotype / Trait</label>
        <input type="text" id="phenotype" placeholder="e.g., Type 2 Diabetes, Height" required>
      </div>
      <br>
      <div class="form-group">
        <label for="ancestry">Population / Ancestry</label>
        <select id="ancestry">
          <option value="EUR">European (EUR)</option>
          <option value="EAS">East Asian (EAS)</option>
          <option value="SAS">South Asian (SAS)</option>
          <option value="AFR">African (AFR)</option>
          <option value="AMR">Admixed American (AMR)</option>
          <option value="Multi">Multi-ancestry</option>
        </select>
      </div>
    </div>
    <!-- 2. Genomic Context -->
    <div class="card">
      <h3>2. Genomic Context</h3>
      <div class="form-group">
        <label for="snpId">Variant ID (Optional)</label>
        <input type="text" id="snpId" placeholder="e.g., rs7903146">
      </div>
      <br>
      <div class="form-group">
        <label for="gene">Target Gene (Optional)</label>
        <input type="text" id="gene" placeholder="e.g., TCF7L2">
      </div>
      <br>
      <div class="form-group">
        <label for="region">Genomic Region</label>
        <input type="text" id="region" placeholder="e.g., 10:114500000-115000000">
      </div>
    </div>
  </div>
</form>

```

```

</div>

<!-- 3. Filters & Quality -->
<div class="card">
  <h3>3. Quality Filters</h3>
  <div class="form-group">
    <label for="pval">P-value Threshold</label>
    <select id="pval">
      <option value="5e-8">Genome-wide sig (< 5e-8)</option>
      <option value="1e-5">Suggestive (< 1e-5)</option>
      <option value="0.05">Nominal (< 0.05)</option>
    </select>
  </div>
  <br>
  <div class="form-group">
    <label for="maf">Min. Allele Frequency (MAF)</label>
    <input type="number" id="maf" value="0.01" step="0.01" min="0" max="0.5">
  </div>
  <br>
  <div class="form-group">
    <label for="info">Min. Info Score</label>
    <input type="number" id="info" value="0.8" step="0.1" min="0" max="1">
  </div>
</div>
</div>

  <button type="submit" id="generateBtn" class="btn">Fetch Data</button>
</form>

<div class="loader" id="loader"></div>

<!-- Results Section -->
<div id="resultsArea">
  <hr style="margin: 30px 0; border-top: 1px solid #ddd;">

  <div class="tabs">
    <button class="tab-btn active" onclick="openTab('summaryStats')">Summary Statistics</button>
    <button class="tab-btn" onclick="openTab('phenotypes')">Phenotype File</button>
    <button class="tab-btn" onclick="openTab('metadata')">Metadata & Logs</button>
  </div>

  <!-- Tab 1: Summary Stats -->
  <div id="summaryStats" class="tab-content active">
    <div style="display:flex; justify-content:space-between; align-items:center;">
      <h3>GWAS Summary Statistics</h3>
      <button class="btn btn-download" onclick="downloadData('sumstats',
'gwas_sumstats.tsv')">Download .TSV</button>
    </div>
    <div style="overflow-x: auto;">
      <table id="sumStatsTable">

```

```

        <thead>
          <tr>
            <th>CHR</th>
            <th>SNP</th>
            <th>BP</th>
            <th>A1 (REF)</th>
            <th>A2 (ALT)</th>
            <th>FRQ_A1</th>
            <th>BETA</th>
            <th>SE</th>
            <th>P</th>
            <th>N</th>
            <th>INFO</th>
          </tr>
        </thead>
        <tbody><!-- JS Populates --></tbody>
      </table>
    </div>
  </div>

  <!-- Tab 2: Phenotypes -->
  <div id="phenotypes" class="tab-content">
    <div style="display:flex; justify-content:space-between; align-items:center;">
      <h3>Individual-Level Phenotype & Covariates</h3>
      <button class="btn btn-download" onclick="downloadData('phe',
'phenotypes.phe')">Download .PHE</button>
    </div>
    <p style="font-size: 0.8rem; color: #666;">Simulated sample subset </p>
    <table id="pheTable">
      <thead>
        <tr>
          <th>FID</th>
          <th>IID</th>
          <th>PHE (Trait)</th>
          <th>SEX</th>
          <th>AGE</th>
          <th>PC1</th>
          <th>PC2</th>
          <th>Genotyping_Batch</th>
        </tr>
      </thead>
      <tbody><!-- JS Populates --></tbody>
    </table>
  </div>

  <!-- Tab 3: Metadata -->
  <div id="metadata" class="tab-content">
    <h3>Study Metadata & Protocol Info</h3>
    <div class="metadata-box" id="metadataContent"></div>
  </div>

```

```

    </div>
</div>

<script>
  // --- Configuration & Constants ---
  const API_URLS = {
    deepseek: "https://api.deepseek.com/v1/chat/completions",
    openai: "https://api.openai.com/v1/chat/completions",
    gemini: "https://generativelanguage.googleapis.com/v1beta/models/gemini-pro:generateContent"
  };

  // State to store generated data for downloading
  let generatedData = {
    summaryStats: [], // Array of objects
    phenotypes: [], // Array of objects
    metadata: ""
  };

  // --- UI Interaction ---
  function openTab(tabName) {
    document.querySelectorAll('.tab-content').forEach(t => t.classList.remove('active'));
    document.querySelectorAll('.tab-btn').forEach(b => b.classList.remove('active'));

    document.getElementById(tabName).classList.add('active');
    // Find button corresponding to tab and make active (simple logic based on order or text is brittle, matching by onclick
    is better)
    const buttons = document.querySelectorAll('.tab-btn');
    if(tabName === 'summaryStats') buttons[0].classList.add('active');
    if(tabName === 'phenotypes') buttons[1].classList.add('active');
    if(tabName === 'metadata') buttons[2].classList.add('active');
  }

  // --- Main Logic ---
  document.getElementById('gwasForm').addEventListener('submit', async function(e) {
    e.preventDefault();

    const provider = document.getElementById('apiProvider').value;
    const apiKey = document.getElementById('apiKey').value;
    const loader = document.getElementById('loader');
    const resultsArea = document.getElementById('resultsArea');

    if (!apiKey) {
      alert('Please enter an API Key.');
```

```

    snpId: document.getElementById('snpId').value || "Significant SNPs",
    region: document.getElementById('region').value || "Genome-wide",
    gene: document.getElementById('gene').value || "Relevant Genes",
    pval: document.getElementById('pval').value,
    maf: document.getElementById('maf').value,
    info: document.getElementById('info').value
  };

// 2. Construct System Prompt (Strict JSON Instruction)
const systemPrompt = `
You are a specialized Bioinformatics Data Generator. Your task is to generate REALISTIC SYNTHETIC GWAS
DATA based on user inputs.
You MUST return the result as a VALID JSON OBJECT with no markdown formatting (no \\`\\`json wrappers).

The JSON object must have three keys: "summary_stats", "phenotype_data", and "metadata".

1. "summary_stats": An array of objects. Each object represents a SNP row with keys:
  - CHR (Chromosome number)
  - SNP (rsID, e.g., rs12345)
  - BP (Position, GRCh37)
  - A1 (Reference Allele)
  - A2 (Effect/Alt Allele)
  - FRQ_A1 (Frequency of A1, float)
  - BETA (Effect size, float)
  - SE (Standard Error, float)
  - P (P-value, scientific notation string e.g., "4.5e-9")
  - N (Sample size, int)
  - INFO (Imputation score, float)
  *Generate about 10 rows. Ensure values make biological sense for the trait requested.*

2. "phenotype_data": An array of objects representing individual samples (generate about 10 rows). Keys:
  - FID (Family ID)
  - IID (Individual ID)
  - PHE (Phenotype value: 0/1 for binary, continuous for quant)
  - SEX (1=Male, 2=Female)
  - AGE (Int)
  - PC1 (Principal Component 1, float)
  - PC2 (Principal Component 2, float)
  - BATCH (Genotyping batch string)

3. "metadata": A string containing specific details:
  - Genotyping Platform (e.g., Illumina GSA)
  - Quality Control Protocols
  - Genome Build version
  - Notes on population structure.
`;

const userPrompt = `
Generate GWAS data for:
Trait: ${inputs.phenotype}

```

Population: \${inputs.ancestry}
 Context: \${inputs.snpId}, Gene: \${inputs.gene}, Region: \${inputs.region}
 Filters: P < \${inputs.pval}, MAF > \${inputs.maf}, Info > \${inputs.info}.

Strictly follow the JSON format instructions.

```

`);

// 3. API Call
loader.style.display = 'block';
resultsArea.style.display = 'none';
document.getElementById('generateBtn').disabled = true;

try {
  let data = null;

  if (provider === 'gemini') {
    data = await callGemini(apiKey, systemPrompt + "\n" + userPrompt);
  } else {
    // OpenAI and DeepSeek share similar structure
    data = await callOpenAICompatible(provider, apiKey, systemPrompt, userPrompt);
  }

  // 4. Parse and Render
  processAndRender(data);

} catch (error) {
  console.error(error);
  alert('Error fetching data: ' + error.message);
} finally {
  loader.style.display = 'none';
  document.getElementById('generateBtn').disabled = false;
}
});

// --- Provider Specific Functions ---

async function callOpenAICompatible(provider, key, sysPrompt, usrPrompt) {
  const url = API_URLS[provider];
  const model = provider === 'deepseek' ? 'deepseek-chat' : 'gpt-3.5-turbo'; // or gpt-4

  const response = await fetch(url, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${key}`
    },
    body: JSON.stringify({
      model: model,
      messages: [
        { role: "system", content: sysPrompt },

```

```

        { role: "user", content: usrPrompt }
      ],
      temperature: 0.5,
      max_tokens: 2500
    })
  });

  if (!response.ok) {
    const err = await response.json();
    throw new Error(err.error?.message || 'API request failed');
  }

  const json = await response.json();
  let content = json.choices[0].message.content;
  return cleanJsonString(content);
}

async function callGemini(key, fullPrompt) {
  const url = `${API_URLS.gemini}?key=${key}`;

  const response = await fetch(url, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      contents: [{ parts: [{ text: fullPrompt }] }]
    })
  });

  if (!response.ok) {
    throw new Error('Gemini API request failed');
  }

  const json = await response.json();
  let content = json.candidates[0].content.parts[0].text;
  return cleanJsonString(content);
}

// Helper to remove markdown code blocks if the AI acts up
function cleanJsonString(str) {
  str = str.replace(/``json/g, "").replace(/``/g, "").trim();
  try {
    return JSON.parse(str);
  } catch (e) {
    console.error("Raw output:", str);
    throw new Error("Failed to parse AI response as JSON. Check console for raw output.");
  }
}

// --- Rendering Logic ---

```

```

function processAndRender(data) {
  // Store globally for downloads
  generatedData.summaryStats = data.summary_stats || [];
  generatedData.phenotypes = data.phenotype_data || [];
  generatedData.metadata = data.metadata || "No metadata provided.";

  // 1. Render Summary Stats
  const ssTable = document.querySelector("#sumStatsTable tbody");
  ssTable.innerHTML = "";
  generatedData.summaryStats.forEach(row => {
    const tr = document.createElement('tr');
    tr.innerHTML = `
      <td>${row.CHR}</td>
      <td>${row.SNP}</td>
      <td>${row.BP}</td>
      <td>${row.A1}</td>
      <td>${row.A2}</td>
      <td>${row.FRQ_A1}</td>
      <td>${row.BETA}</td>
      <td>${row.SE}</td>
      <td style="color:${parseFloat(row.P) < 5e-8 ? 'red' : 'black'}; font-weight:bold;">${row.P}</td>
      <td>${row.N}</td>
      <td>${row.INFO}</td>
    `;
    ssTable.appendChild(tr);
  });

  // 2. Render Phenotypes
  const pheTable = document.querySelector("#pheTable tbody");
  pheTable.innerHTML = "";
  generatedData.phenotypes.forEach(row => {
    const tr = document.createElement('tr');
    tr.innerHTML = `
      <td>${row.FID}</td>
      <td>${row.IID}</td>
      <td>${row.PHE}</td>
      <td>${row.SEX}</td>
      <td>${row.AGE}</td>
      <td>${row.PC1}</td>
      <td>${row.PC2}</td>
      <td>${row.BATCH}</td>
    `;
    pheTable.appendChild(tr);
  });

  // 3. Render Metadata
  document.getElementById('metadataContent').textContent = generatedData.metadata;

  // Show results
  document.getElementById('resultsArea').style.display = 'block';
}

```

```

}

// --- Download Logic ---
function downloadData(type, filename) {
  let content = "";

  if (type === 'sumstats') {
    // Header
    content += "CHR\tSNP\tBP\tA1\tA2\tFRQ\tBETA\tSE\tP\tN\tINFO\n";
    // Rows
    generatedData.summaryStats.forEach(row => {
      content +=
`$${row.CHR}\t${row.SNP}\t${row.BP}\t${row.A1}\t${row.A2}\t${row.FRQ_A1}\t${row.BETA}\t${row.SE}\t${row.P}\t${r
ow.N}\t${row.INFO}\n`;
    });
  } else if (type === 'phe') {
    // Header
    content += "FID\tIID\tPHE\tSEX\tAGE\tPC1\tPC2\tBATCH\n";
    // Rows
    generatedData.phenotypes.forEach(row => {
      content +=
`$${row.FID}\t${row.IID}\t${row.PHE}\t${row.SEX}\t${row.AGE}\t${row.PC1}\t${row.PC2}\t${row.BATCH}\n`;
    });
  }
}

const blob = new Blob([content], { type: 'text/tab-separated-values' });
const url = window.URL.createObjectURL(blob);
const a = document.createElement('a');
a.href = url;
a.download = filename;
document.body.appendChild(a);
a.click();
document.body.removeChild(a);
}
</script>
</body>
</html>

```

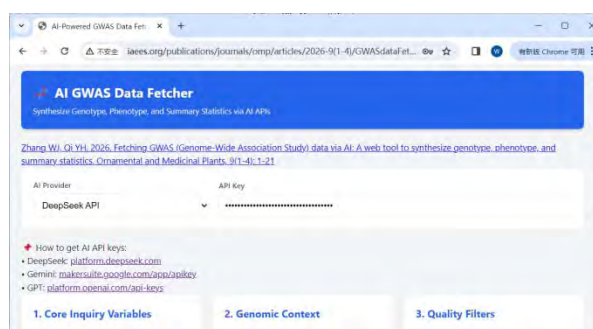


Fig. 1 The web tool for fetching GWAS data via AI.

5 Demo

In present demo, the trait is chosen as Type 2 Diabetes; the population is Multi-ancestry; use DeepSeek for AI data fetching. Before using this tool, we need to register to obtain an AI API key. For example, for DeepSeek, log in to <https://platform.deepseek.com/>, register an account, and prepay a very small fee to use the assigned API key. The GWAS summary statistics are fetched as follows:

| CHR | SNP | BP | A1 | A2 | FRQ | BETA | SE | P | N | INFO |
|-----|------------|-----------|----|----|------|------|------|----------|-------|------|
| 10 | rs7903146 | 114758349 | C | T | 0.31 | 0.18 | 0.03 | 3.20E-09 | 75000 | 0.95 |
| 11 | rs5219 | 17408671 | T | C | 0.35 | 0.12 | 0.02 | 2.10E-08 | 75000 | 0.92 |
| 3 | rs1801282 | 12393125 | G | C | 0.85 | 0.15 | 0.03 | 1.50E-09 | 75000 | 0.94 |
| 1 | rs340874 | 118544779 | C | T | 0.42 | 0.09 | 0.02 | 4.30E-08 | 75000 | 0.91 |
| 16 | rs9939609 | 53786615 | A | T | 0.4 | 0.11 | 0.02 | 2.80E-08 | 75000 | 0.93 |
| 6 | rs9465871 | 20606539 | C | G | 0.28 | 0.14 | 0.03 | 1.90E-08 | 75000 | 0.89 |
| 2 | rs7578597 | 227101796 | T | C | 0.12 | 0.2 | 0.04 | 3.70E-09 | 75000 | 0.88 |
| 8 | rs13266634 | 118184782 | C | T | 0.65 | 0.1 | 0.02 | 4.10E-08 | 75000 | 0.92 |
| 17 | rs4430796 | 36082825 | G | A | 0.38 | 0.13 | 0.03 | 2.50E-08 | 75000 | 0.9 |
| 9 | rs10811661 | 22125504 | T | C | 0.47 | 0.11 | 0.02 | 3.90E-08 | 75000 | 0.94 |

The following are the fetched information for individual-level phenotype and covariates:

| FID | IID | PHE | SEX | AGE | PC1 | PC2 | BATCH |
|--------|--------|-----|-----|-----|--------|--------|---------|
| FAM001 | IND001 | 1 | 1 | 58 | -0.012 | 0.005 | BATCH_A |
| FAM002 | IND002 | 0 | 2 | 45 | 0.008 | -0.003 | BATCH_B |
| FAM003 | IND003 | 1 | 1 | 62 | -0.015 | 0.01 | BATCH_A |
| FAM004 | IND004 | 0 | 2 | 51 | 0.006 | -0.007 | BATCH_C |
| FAM005 | IND005 | 1 | 1 | 67 | -0.02 | 0.012 | BATCH_B |
| FAM006 | IND006 | 0 | 2 | 48 | 0.004 | -0.009 | BATCH_A |
| FAM007 | IND007 | 1 | 1 | 55 | -0.018 | 0.008 | BATCH_C |
| FAM008 | IND008 | 0 | 2 | 53 | 0.01 | -0.005 | BATCH_B |
| FAM009 | IND009 | 1 | 1 | 60 | -0.014 | 0.006 | BATCH_A |
| FAM010 | IND010 | 0 | 2 | 49 | 0.007 | -0.004 | BATCH_C |

The following are the produced study metadata and protocol information:

Genotyping Platform: Illumina Global Screening Array v3.0. Quality Control Protocols: Standard GWAS QC applied: sample call rate > 98%, SNP call rate > 95%, HWE $p > 1e-6$, MAF > 0.01, INFO > 0.8. Genome Build version: GRCh37/hg19. Notes on population structure: Multi-ancestry cohort (European, East Asian, African) with principal components computed to adjust for stratification; top PCs included as covariates in association analysis.

6 Discussion

The core principle for fetching GWAS data via AI is leveraging natural language processing (NLP) and knowledge retrieval capabilities to access human-readable summaries of GWAS data and then structure that information, rather than direct database scraping from public databases. AI acts as a powerful intelligent agent that searches, comprehends, and synthesizes already-published and curated GWAS results. The procedures and tasks for AI GWAS data fetching are (Zhang, 2026a):

(1) Data Source Identification and Access

The data sources are publicly available, authoritative databases and scientific literature as:

GWAS Catalog: The central, curated repository of published GWAS findings. AI can query it based on traits, genes, SNPs, or p-values.

PubMed / Europe PMC: AI searches through millions of scientific articles to find specific GWAS studies, their methodologies, and full results.

Other specialized databases: This includes resources like the NHGRI-EBI GWAS Catalog, Open Targets Genetics, and GTEx Portal for functional annotations.

(2) Natural Language Processing (NLP) and Information Extraction

This is the core of the "retrieval" process. When AI accesses a scientific paper or a database entry, it doesn't just copy text but:

Named Entity Recognition (NER): Identify and extract key biological entities:

SNPs: (e.g., `rs429358`, `rs7412`)

Genes: (e.g., `APOE`, `HLA-DRB1`)

Phenotypes/Traits: (e.g., "Alzheimer's disease," "Height," "Type 2 Diabetes").

Relationship extraction: Understand the connections between these entities.

Example: From the sentence "The SNP rs429358 in the APOE gene was significantly associated with Alzheimer's disease (p-value = 5.0e-300)", AI extracts the triplet: `(rs429358, associated with, Alzheimer's disease)`.

Statistical data extraction: Parse complex numerical and statistical information.

P-values, Odds Ratios (OR), Beta coefficients, Confidence Intervals (CI), and Effect Allele Frequencies.

(3) Data Structuring and Integration

The extracted information, which is often unstructured in text, is organized into a structured format that is easy for the user to understand and use, such as a markdown table:

(4) Knowledge Synthesis and Reasoning

AI can go beyond simple retrieval by connecting information from multiple sources:

Cross-referencing: Finding all GWAS hits for a specific gene across different studies.

Functional enrichment: Linking significant SNPs to their potential regulatory roles (e.g., by connecting to eQTL data from GTEx).

Summarization: Providing a concise summary of the genetic architecture of a complex trait based on the top findings.

Important limitations of AI include:

Access individual-level data: AI cannot retrieve the raw, individual genotype or phenotype data from controlled-access databases like dbGaP. This requires specific authorizations and approvals.

Perform original GWAS analysis: AI do not run new statistical genetics analyses on raw data.

Guarantee 100% completeness: While AI aims for high accuracy, the retrieval is dependent on the source data being up-to-date and correctly parsed. It's always good practice to verify critical findings in the original source.

In conclusion, the principle of AI fetching of GWAS data is not one of low-level data scraping, but of high-level, intelligent information retrieval and synthesis. It uses advanced NLP to read, understand, and organize the vast and complex findings from the GWAS literature and public databases, making them easily accessible to the user in a structured and meaningful way. We should always keep in mind to verify critical findings in the original source for updating fetched data.

References

- 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/>
- 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](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](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](http://www.iaees.org/publications/journals/np/articles/2026-11(3-4)/1-Zhang-Abstract.asp)
- Zhang WJ. 2026b. A multi-source GWAS data fetcher for Mendelian Randomization analysis. *Network Pharmacology*, 11(1-2): 1-61.
[http://www.iaees.org/publications/journals/np/articles/2026-11\(1-2\)/1-Zhang-Abstract.asp](http://www.iaees.org/publications/journals/np/articles/2026-11(1-2)/1-Zhang-Abstract.asp)