GeoNLI

AI-Powered Satellite Image Analysis Platform

User Guide

Version	1.0.0
Release Date	December 2025
Platform	Web Application
Backend	FastAPI + Python
Frontend	React 19

Table of Contents

1. Introduction

1.1 What is GeoNLI?

GeoNLI is an advanced AI-powered platform for analyzing satellite imagery through natural language interactions. It combines state-of-the-art computer vision models with an intuitive chat interface to provide detailed image analysis, object detection, and visual question answering.

The platform leverages three specialized AI services:

Service	Model	Purpose
Caption	Florence-2	Generate detailed scene descriptions
VQA	Florence-2	Answer questions about images
Grounding	GeoGround + YOLO	Detect objects with precise bounding boxes

1.2 Key Features

- **Intelligent Image Analysis:** Upload satellite images and receive detailed AI-generated captions
- **Natural Language Queries:** Ask questions about your images in plain English
- **Object Detection:** Automatically identify and locate objects with oriented bounding boxes
- **Session Management:** Organize your analyses into sessions for easy reference
- **Export Capabilities:** Download analysis results and chat conversations
- **Multi-user Support:** Secure authentication with personal workspaces
- **Responsive Interface:** Works seamlessly on desktop and tablet devices
- **Real-time Processing:** Get instant AI responses as you interact

1.3 System Requirements

For Users (Web Interface):

- Modern web browser (Chrome, Firefox, Safari, or Edge)
- Stable internet connection (minimum 5 Mbps)

• Screen resolution: 1280x720 or higher

For Deployment (Backend):

• Python 3.10 or 3.11

• 16GB RAM minimum (32GB recommended)

• 50GB storage (100GB recommended)

• NVIDIA GPU with 16GB+ VRAM (optional but recommended)

• MongoDB 4.4+ (local or Atlas)

2. Quick Start Guide

This section will help you get started with using GeoNLI in just a few minutes. If you're setting up the application for the first time, please refer to Sections 3 and 4 for deployment instructions.

2.1 Account Creation

Step 1: Navigate to the GeoNLI application URL

```
https://your-geonli-instance.com
```

Step 2: Click on the 'Sign Up' tab

Step 3: Fill in your registration details:

• Full Name: Your complete name

• Email: A valid email address

• Password: Minimum 6 characters

Step 4: Click 'Create Account'

Note: Upon successful registration, you'll be automatically logged in.

2.2 First Login

After creating your account or for subsequent visits:

Step 1: Go to the login page

Step 2: Enter your credentials:

• Email: Your registered email

• Password: Your account password

Step 3: Click 'Sign In'

You'll be redirected to the main dashboard with three panels:

• **Left Panel:** Session list and management

• **Center Panel:** Image workspace

• **Right Panel:** AI chat console

2.3 Your First Analysis

Step 1: Upload an Image

In the center panel (Image Workspace):

1. Click the upload area or drag and drop an image

2. Select a satellite image (JPEG or PNG format)

3. Wait for the upload to complete

Step 2: Ask Questions

In the right panel (Chat Console), type your first question:

```
What can you see in this satellite image?
```

Press Enter or click the send button.

Step 3: View Results

The AI will analyze your image and respond with:

• A detailed caption describing the scene

• Answers to your specific questions

• Object detection results if requested

Step 4: Continue the Conversation

You can ask follow-up questions such as:

• "How many buildings are visible?"

• "What color are the roofs?"

• "Can you locate all the vehicles?"

Tip: Each session maintains conversation history, allowing the AI to understand context.

3. Backend Deployment

This section provides detailed instructions for deploying the GeoNLI backend, which powers all AI analysis capabilities. You can deploy locally for development or to Modal for production.

3.1 Prerequisites

Software Requirements:

Software	Version	Purpose
Python	3.10 or 3.11	Backend runtime
MongoDB	4.4+	Database storage
Git	Latest	Version control
CUDA	12.1+ (optional)	GPU acceleration

Model Files:

You'll need three pre-packaged model environment files:

• **captioning.zip** (~2-3GB) - Florence-2 caption model

• **vqa.zip** (~2-3GB) - Florence-2 VQA model

• **grounding.zip** (~15-20GB) - GeoGround + YOLO models

Warning: Ensure you have sufficient storage space (~20GB minimum) before downloading.

3.2 Local Deployment

Step 1: Extract the Deployment Package

Extract the provided `multi-model-env-backend.zip` file:

```
unzip multi-model-env-backend.zip
cd multi-model-env-backend
```

Step 2: Download Model Files

If model files are not included in the package, download them from the provided Google Drive links and place them in the `multi-model-env-backend/` directory.

```
pip install gdown
gdown <CAPTIONING_GOOGLE_DRIVE_LINK>
gdown <VQA_GOOGLE_DRIVE_LINK>
gdown <GROUNDING_GOOGLE_DRIVE_LINK>
```

Step 3: Create Virtual Environment

```
python3.11 -m venv venv
source venv/bin/activate # On Linux/Mac
# OR
venv\Scripts\activate # On Windows
```

Step 4: Install Dependencies

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

This will install all required Python packages including FastAPI, PyTorch, and model dependencies.

Step 5: Extract Model Environments

```
unzip captioning.zip -d captioning_env/
unzip vqa.zip -d vqa_env/
unzip grounding.zip -d grounding_env/
```

Note: This step may take several minutes due to large file sizes.

Step 6: Configure MongoDB

Option A: Use MongoDB Atlas (Recommended)

1. Go to https://www.mongodb.com/cloud/atlas

2. Create a free account and cluster

3. Create a database user with password

4. Whitelist IP address: Add 0.0.0.0/0 for development

5. Get connection string from 'Connect' button

Option B: Install MongoDB Locally

```
# Ubuntu/Debian
sudo apt-get install mongodb-community

# macOS
brew install mongodb-community

# Start service
sudo systemctl start mongodb
```

Step 7: Create Configuration File

Create a `.env` file in the project root with the following content:

```
# Database
MONGODB_URL=mongodb+srv://username:password@cluster.mongodb.net/
DATABASE_NAME=geonli_db

# Authentication
JWT_SECRET=your-generated-secret-key
JWT_ALGORITHM=HS256
ACCESS_TOKEN_EXPIRE_DAYS=7

# Server
HOST=0.0.0.0
PORT=8000
NODE_ENV=development
FRONTEND_URL=http://localhost:5173

# Model Paths
CAPTION_MODEL_PATH=./checkpoints/caption_model
VQA_MODEL_PATH=./checkpoints/vqa_model
GROUNDING_MODEL_PATH=./models/geoground
```

Important: Generate a secure JWT secret using: `openssl rand -hex 32`

Step 8: Start the Backend Server

```
python src/server.py
```

The server will start on http://localhost:8000. You should see:

```
■ Starting GeoNLI Backend...
■ Environment: Local
■ Database initialized
INFO: Uvicorn running on http://0.0.0.0:8000
```

Step 9: Verify Installation

Test the health endpoint:

```
curl http://localhost:8000/health
```

Visit the API documentation at: http://localhost:8000/docs

3.3 Cloud Deployment (Modal)

Modal provides serverless GPU infrastructure ideal for AI workloads. This deployment option offers automatic scaling and pay-per-use pricing.

Step 1: Install Modal CLI

```
pip install modal
```

Step 2: Authenticate with Modal

```
modal token new
```

Follow the prompts to log in to your Modal account.

Step 3: Configure Modal Secrets

Store your environment variables securely in Modal:

```
modal secret create geonli-secrets \
MONGODB_URL="mongodb+srv://username:password@cluster.mongodb.net/" \
DATABASE_NAME="geonli_db" \
JWT_SECRET="your-jwt-secret" \
FRONTEND_URL="https://your-frontend-domain.com"
```

Step 4: Prepare Deployment Files

Ensure these files are in your project root:

• modal_app.py (deployment configuration)

• captioning.zip

• vqa.zip

• grounding.zip

• src/ directory (backend source code)

Note: For Modal deployment, keep zip files compressed - Modal extracts them automatically.

Step 5: Deploy to Modal

```
modal deploy modal_app.py
```

Modal will build Docker images, upload your files, and deploy the application. You'll receive URLs for your deployed endpoints:

```
✓ App deployed! ■

Web endpoints:
■■ backend: https://username--multi-model-env-backend-backend.modal.run
■■ health: https://username--multi-model-env-backend-health.modal.run
■■ router: https://username--multi-model-env-backend-router.modal.run
```

Important: Save these URLs - you'll need the backend URL for frontend configuration.

Step 6: Test Deployment

```
curl https://<username>--multi-model-env-backend-health.modal.run
```

Step 7: Monitor Logs

View real-time logs:

```
modal app logs multi-model-env-backend --follow
```

Deployment Tips:

• First deployment takes 5-10 minutes as Modal builds images

• Subsequent deploys are faster (2-3 minutes)

• Cold starts take 30-60 seconds, warm requests are instant

• Modal automatically scales based on demand

4. Frontend Setup

The GeoNLI frontend is a React 19 application that provides the user interface. This section guides you through setting up and running the frontend.

4.1 Installation

Step 1: Extract Frontend Package

```
unzip frontend.zip
cd frontend
```

Step 2: Verify Node.js Installation

Ensure you have Node.js v20.19.0 or v22.12.0+:

```
node --version
npm --version
```

If Node.js is not installed, download from: https://nodejs.org/

Step 3: Install Dependencies

```
npm install
```

This installs all required packages including React, Vite, Tailwind CSS, and Zustand.

4.2 Configuration

Create Environment File

Create a `.env` file in the frontend directory:

```
# For local backend
VITE_IPV4_URL=http://localhost:8000

# For Modal backend
VITE_BACKEND_URL=https://username--multi-model-env-backend-backend.modal.run

# Optional: Custom port
VITE_PORT=5173
```

Important: Replace the Modal URL with your actual deployment URL from Section 3.3.

4.3 Running Development Server

Start the Development Server

```
npm run dev
```

The application will start on http://localhost:5173. You should see:

```
VITE v7.2.4 ready in 500 ms

➜ Local: http://localhost:5173/
➜ Network: http://192.168.1.100:5173/
```

Open your web browser and navigate to http://localhost:5173 to access the application.

Production Build

To create a production build:

```
npm run build
```

The optimized build will be created in the `dist/` directory. You can deploy this to any static hosting service (Vercel, Netlify, etc.).

5. Using the Application

5.1 Dashboard Overview

The GeoNLI dashboard consists of three main panels:

Panel	Location	Purpose
Session List	Left	View and manage analysis sessions
Image Workspace	Center	Upload and view satellite images
Chat Console	Right	Interact with AI for image analysis

Header Bar Features:

• **Theme Toggle:** Switch between dark and light modes

• **Share Button:** Share the current session

• **Profile Menu:** Access account settings and logout

Panel Resizing:

You can resize the left and right panels by dragging the divider between panels. This allows you to customize your workspace based on your needs.

5.2 Uploading Images

Supported Formats:

• JPEG (.jpg, .jpeg)

• PNG (.png)

Upload Methods:

Method 1: Drag and Drop

1. Locate your satellite image file

2. Drag the file to the Image Workspace (center panel)

3. Drop the file when the upload area is highlighted

Method 2: Click to Upload

1. Click the upload area in the Image Workspace

2. Browse and select your image file

3. Click 'Open' to confirm

After Upload:

• The image will display in the center panel

• Zoom controls appear in the top-right corner

• A download button allows you to save the image

Tip: You can zoom in/out using the +/- buttons or reset to 100% zoom.

5.3 AI Chat Interface

The AI chat console on the right allows you to interact with the AI about your uploaded images.

Asking Questions:

Step 1: Type your question in the input field at the bottom

Step 2: Press Enter or click the send button

Step 3: Wait for the AI to analyze and respond

Example Questions:

• "What objects can you identify in this image?"

• "Describe the overall scene in detail"

• "How many buildings are visible?"

• "What is the predominant land use type?"

• "Can you locate all the vehicles?"

• "What color are the roofs of the buildings?"

• "Are there any water bodies present?"

• "Estimate the size of the largest structure"

Message Actions:

• **Copy:** Click the copy icon to copy a message to clipboard

• **Export:** Click 'Export Chat' to download the conversation as a text file

Note: The AI maintains context within a session, so you can ask follow-up questions.

5.4 Session Management

Sessions help you organize your analyses. Each session maintains its own chat history and uploaded images.

Creating a New Session:

1. Click the '+ New Chat' button in the left panel

2. A new session is created with a default name

3. The session becomes active immediately

Renaming a Session:

1. Hover over a session in the list

2. Click the three-dot menu icon

3. Select 'Rename'

4. Enter the new name and confirm

Archiving a Session:

1. Open the session menu (three dots)

2. Select 'Archive'

3. Archived sessions are hidden from the main list

4. Toggle 'Show Archived' to view archived sessions

Deleting a Session:

1. Open the session menu

2. Select 'Delete'

3. Confirm the deletion

Warning: Deleting a session permanently removes all associated messages and images.

Searching Sessions:

1. Click the search icon in the left panel

2. Type keywords to filter sessions by name

3. Click the X icon to clear the search

5.5 Advanced Features

Object Detection (Grounding):

To get precise object locations with bounding boxes:

1. Upload an image

2. Ask: "Can you locate all [objects] in this image?"

3. Example: "Can you locate all storage tanks?"

4. The AI will provide coordinates for each detected object

Detailed Captions:

Ask: "Generate a detailed caption for this image"

The AI will provide a comprehensive description including:

• Scene composition

• Major features and structures

• Colors and textures

• Spatial relationships

Visual Question Answering (VQA):

Ask specific questions about image attributes:

• Binary: 'Is there a runway in this image?'

• Numeric: 'How many cars are in the parking lot?'

• Semantic: 'What is the primary color of the buildings?'

Exporting Results:

Export Chat:

1. Click the 'Export Chat' button at the bottom of the chat panel

2. A .txt file will be downloaded with the complete conversation

3. The filename includes the session name and timestamp

Download Image:

1. Click the download icon in the Image Workspace

2. The current image will be saved to your downloads folder

Profile Settings:

Access your profile by clicking your avatar in the top-right:

• Update your name and profile picture

• Change your password

• View account statistics (sessions, images analyzed)

• Manage preferences (theme, notifications)

Keyboard Shortcuts:

Action	Shortcut
Send message	Enter
New line in message	Shift + Enter
Focus search	Ctrl + F
New session	Ctrl + N

6. API Reference

This section provides technical details for developers integrating with the GeoNLI API.

6.1 GeoNLI Evaluation Endpoint

Endpoint:

```
POST /geoNLI/eval
```

Description:

Performs comprehensive satellite image analysis including caption generation, object detection, and visual question answering.

Request Format:

```
{
"input_image": {
"image_id": "sample1.png",
"image_url": "https://example.com/image.png",
"metadata": {
"width": 512,
"height": 512,
"spatial_resolution_m": 1.57
}
},
"queries": {
"caption_query": {
"instruction": "Generate a detailed caption"
},
"grounding_query": {
"instruction": "Locate all storage tanks"
},
"attribute_query": {
"binary": {"instruction": "Are there any digits visible?"},
"numeric": {"instruction": "How many storage tanks?"},
"semantic": {"instruction": "What color are the tanks?"}
}
}
}
```

Response Format:

Returns the same structure with 'response' fields populated for each query.

```
{
"input_image": { ... },
"queries": {
"caption_query": {
"instruction": "...",
"response": "The satellite image shows..."
},
"grounding_query": {
"instruction": "...",
"response": [
{"object-id": "1", "obbox": [x1,y1,x2,y2,x3,y3,x4,y4]}
]
},
"attribute_query": {
"binary": {"instruction": "...", "response": "Yes"},
"numeric": {"instruction": "...", "response": 8.0},
"semantic": {"instruction": "...", "response": "White"}
}
}
}
```

Note: All queries are optional. You can include any combination based on your needs.

6.2 Authentication

Sign Up:

```
POST /api/auth/signup

Request Body:
{"email": "user@example.com", "fullName": "John Doe", "password": "pass123"}

Response: User object with JWT cookie set
```

Login:

```
POST /api/auth/login

Request Body:
{"email": "user@example.com", "password": "pass123"}

Response: User object with JWT cookie set
```

Check Auth:

```
GET /api/auth/check

Headers: Cookie: jwt=<token>

Response: Current user object
```

Logout:

```
POST /api/auth/logout

Headers: Cookie: jwt=<token>

Response: Success message, JWT cookie cleared
```

6.3 Session Management

Create Session:

```
POST /api/sessions

Request Body:
{"name": "My Analysis", "archived": false, "projectId": null}
```

Get All Sessions:

```
GET /api/sessions

Response: Array of session objects
```

Update Session:

```
PUT /api/sessions/{session_id}

Request Body:
{"name": "Updated Name", "archived": true}
```

Delete Session:

```
DELETE /api/sessions/{session_id}
```

Note: All session endpoints require authentication (JWT cookie).

7. Troubleshooting

7.1 Common Issues

Issue: Cannot access the application

Solution:

• Verify both backend and frontend servers are running

• Check firewall settings allow connections on ports 5173 and 8000

• Ensure correct URLs in .env files

• Try clearing browser cache and cookies

Issue: Login fails with 'Invalid credentials'

Solution:

• Double-check email and password

• Passwords are case-sensitive

• Try resetting your password (if implemented)

• Verify backend database connection is working

Issue: Images won't upload

Solution:

• Check file format (only JPEG and PNG supported)

• Verify file size is under limit (usually 10MB)

• Ensure you have an active session

• Check browser console for error messages

• Verify backend /api/images/upload endpoint is working

Issue: AI responses are slow or timeout

Solution:

• First request after deployment may take 30-60s (cold start)

• Subsequent requests should be faster

• For local deployment, ensure GPU is being utilized

• Check backend logs for errors

• Try with a smaller image

Issue: Dark/Light theme not saving

Solution:

• Check if browser allows localStorage

• Clear browser cache and try again

• Ensure cookies are not blocked

• Try in a different browser

7.2 Backend Problems

MongoDB Connection Error

```
Error: ServerSelectionTimeoutError: connection refused
```

Solutions:

• Verify MONGODB_URL in .env is correct

• Check MongoDB Atlas IP whitelist includes your IP

• For local MongoDB, ensure service is running:

```
sudo systemctl status mongodb
```

• Test connection with mongosh

CUDA Out of Memory

```
RuntimeError: CUDA out of memory
```

Solutions:

• Check GPU memory usage: nvidia-smi

• Close other GPU-intensive applications

• Reduce batch sizes in .env configuration

• Switch to CPU mode (slower but works):

```
export CUDA_VISIBLE_DEVICES=""
```

Model Files Not Found

```
FileNotFoundError: No such file or directory
```

Solutions:

• Verify all three zip files were extracted

• Check model paths in .env match directory structure

• Re-extract zip files if needed

• Ensure sufficient disk space for extraction

Port Already in Use

```
OSError: Address already in use
```

Solutions:

- Find process using port 8000:

  ```
  lsof -i :8000
  ```

- Kill the process:

  ```
  kill -9 <PID>
  ```

- Or use a different port in .env

7.3 Frontend Problems

Blank Screen After Login

Solutions:

• Open browser console (F12) and check for errors

• Verify VITE_BACKEND_URL in .env is correct

• Clear browser cache (Ctrl+Shift+Delete)

• Clear localStorage: Open console and run:

```
localStorage.clear()
```

• Verify backend is accessible from frontend

CORS Errors

```
Access to XMLHttpRequest blocked by CORS policy
```

Solutions:

• Add frontend URL to CORS allowed origins in backend server.py

• Ensure FRONTEND_URL in backend .env matches actual frontend URL

• Restart backend after configuration changes

• For Modal deployment, update CORS settings in deployed app

Build Failures

Solutions:

• Delete node_modules and package-lock.json

• Run npm install again

• Verify Node.js version is compatible (v20+)

• Check for syntax errors in code

• Clear Vite cache:

```
rm -rf dist node_modules/.vite
```

8. Best Practices

Image Quality:

• Use high-resolution satellite images for best results

• Ensure images are clear and not heavily compressed

• Optimal resolution: 512x512 pixels or higher

• Avoid images with excessive noise or artifacts

Asking Questions:

• Be specific in your questions

• Use clear, descriptive language

• Ask one question at a time for best accuracy

• Provide context when asking follow-up questions

• Use object-specific terminology (e.g., 'storage tanks' vs 'round things')

Session Organization:

• Create separate sessions for different analysis tasks

• Use descriptive session names (e.g., 'Urban Area Analysis - January 2025')

• Archive completed sessions to keep workspace clean

• Export important conversations before deleting sessions

• Regularly clean up unused sessions

Performance Optimization:

• Close unused browser tabs to free memory

• Use the latest browser version for best performance

• On slow connections, wait for complete upload before querying

• For batch processing, use the API directly instead of UI

Security:

• Use strong, unique passwords

- Log out when using shared computers
- Don't share your account credentials
- For production deployments, use HTTPS
- Regularly update passwords

Data Management:

- Export important analysis results regularly
- Keep backup copies of original images
- Document your analysis methodology in session names/notes
- Consider data retention policies for sensitive imagery

9. Support & Resources

Documentation:

• Backend README: Comprehensive backend setup guide

• Frontend README: Complete frontend documentation

• API Documentation: Interactive API docs at /docs endpoint

• Modal Deployment Guide: Cloud deployment instructions

Interactive API Documentation:

When the backend is running, access interactive API documentation:

• Swagger UI: http://localhost:8000/docs

• ReDoc: http://localhost:8000/redoc

Example Files:

• example_requests.json - Sample API requests

• integration_example.py - Python integration code

• modal_client.py - Modal API client

• test_modal.sh - Testing script

Technology Stack:

Component	Technology	Version
Backend	FastAPI + Python	0.121.3 / 3.11
Frontend	React + Vite	19.0.0 / 7.2.4
Database	MongoDB	4.4+
AI Models	Florence-2, GeoGround, YOLO	-
State Management	Zustand	5.0.3
Styling	Tailwind CSS	4.1.4
Cloud	Modal (optional)	-

External Resources:

• FastAPI: https://fastapi.tiangolo.com

• React: https://react.dev

• MongoDB Atlas: https://www.mongodb.com/cloud/atlas

• Modal: https://modal.com/docs

• Florence-2: https://huggingface.co/microsoft/Florence-2-large

Need Help?

If you encounter issues not covered in this guide:

1. Check the Troubleshooting section (Section 7)

2. Review backend/frontend README files

3. Check browser console for error messages

4. Review backend logs for detailed error information

5. Consult the interactive API documentation

Appendix: Quick Reference

Common Commands:

Task	Command
Start Backend (Local)	`python src/server.py`
Start Frontend (Dev)	`npm run dev`
Build Frontend	`npm run build`
Deploy to Modal	`modal deploy modal_app.py`
View Modal Logs	`modal app logs multi-model-env-backend --follow`
Test Health	`curl http://localhost:8000/health`
Generate JWT Secret	`openssl rand -hex 32`

Default Ports:

• Backend: 8000

• Frontend: 5173

• MongoDB (local): 27017

Important File Locations:

• Backend .env: multi-model-env-backend/.env

• Frontend .env: frontend/.env

• Backend logs: Terminal output or logs/

• Uploaded images: multi-model-env-backend/uploads/

Deployment Checklist:

■ Python 3.10/3.11 installed

■ Node.js v20+ installed

■ MongoDB configured (Atlas or local)

■ All three model zip files downloaded

■ Backend .env file created and configured

- ■ Frontend .env file created and configured
- ■ JWT secret generated
- ■ Backend dependencies installed (pip install -r requirements.txt)
- ■ Frontend dependencies installed (npm install)
- ■ Model environments extracted (for local) or zipped (for Modal)
- ■ Backend server started and health check passes
- ■ Frontend server started and accessible
- ■ Login/Signup working correctly
- ■ Image upload working
- ■ AI responses working

End of User Guide

Thank you for using GeoNLI! We hope this guide helps you get the most out of the platform.

For the latest updates and additional resources, please refer to the project documentation.