#include <QApplication>

#include <QMainWindow>

#include <QLineEdit>

#include <QTextEdit>

#include <QWebEngineView>

#include <QTabWidget>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QPushButton>

#include <QLabel>

#include <QDockWidget>

#include <QDialog>

#include <QMessageBox>

#include <QToolBar>

#include <QScreen>

#include <QDragEnterEvent>

#include <QDropEvent>

#include <QMimeData>

#include <vtkSmartPointer.h>

#include <vtkScatterPlotMatrix.h>

#include <curl/curl.h>

#include <websocket.h>

#include <sqlite3.h>

#include <aws/core/Aws.h>

#include <aws/s3/S3Client.h>

#include <aws/cognito-idp/CognitoIdentityProviderClient.h>

#include <pocketsphinx.h>

#include <opencv2/opencv.hpp>

#include <pybind11/embed.h>

#include <qalculate.h>

#include <windows.h>

#include <string>

#include <vector>

#include <thread>

#include <nlohmann/json.hpp>

#define MAX\_QUERY\_LENGTH 3000

#define MAX\_WINDOWS 20

#define NASA\_API\_KEY\_1 "PNJaNeFWqMb2g0CEQGqJePkndqYfKvBzq6XJqAwg"

#define NASA\_API\_KEY\_2 "FJnBo64nLFqExHwDchrcaf101D8wmGSm0cF27clz"

#define MAST\_API\_KEY "emXvt90Htf0U4RogKTB5lqSxClUeg2pvMQxvZciM"

#define OPENAI\_API\_KEY "sk-proj-QYkFLINUIg8yy1E-exV9j-Kf7HepwIp6tY-0lDpYzRTfbvVsc4xKpDv2nNvts8Ia\_bK7Sm0z-CT3BlbkFJFR1YHH1SbEZz16GwX1ebPrU2h8GOrmGF1vKm0ECTsGeD\_HnTkRqnaCJgI9\_HFvZxh\_Xvf27MwA"

#define COGNITO\_CLIENT\_ID "your\_cognito\_client\_id"

#define COGNITO\_REGION "us-east-1"

namespace py = pybind11;

using json = nlohmann::json;

// Structure for search results

struct SearchResult {

std::string url;

std::string title;

std::string summary;

double relevance;

bool isLive;

};

// Global variables

std::vector<std::string> focusList = {

"Worldwide Telescopes", "NASA", "SpaceX", "JPL", "ESA", "STScI",

"Hubble", "JWST", "Chandra", "ALMA", "EHT", "SKA Observatory",

"CERN", "DARPA", "ATIP", "ACS Hubble Ultra Deep Field",

"WFC3 Hubble Deep Field", "Hubble Heritage Team", "LIGO", "FAST"

};

std::vector<SearchResult> results[MAX\_WINDOWS];

sqlite3\* db;

Aws::S3::S3Client\* s3\_client;

Aws::CognitoIdentityProvider::CognitoIdentityProviderClient\* cognito\_client;

// Scientific Calculator

class ScientificCalculatorDialog : public QDialog {

public:

ScientificCalculatorDialog(QWidget\* parent) : QDialog(parent) {

setWindowFlags(Qt::Window | Qt::FramelessWindowHint);

setAcceptDrops(true);

QVBoxLayout\* layout = new QVBoxLayout(this);

input = new QTextEdit(this);

input->setPlaceholderText("Enter equations (e.g., d/dx(x^2), ∫(0,1) x^2 dx, x^2 + y = 5)");

input->setMinimumHeight(100);

input->setMaximumHeight(1000);

input->setAcceptDrops(true);

output = new QTextEdit(this);

output->setReadOnly(true);

QPushButton\* solveBtn = new QPushButton("Solve", this);

layout->addWidget(input);

layout->addWidget(solveBtn);

layout->addWidget(output);

connect(solveBtn, &QPushButton::clicked, this, &ScientificCalculatorDialog::solveEquations);

connect(input, &QTextEdit::textChanged, this, &ScientificCalculatorDialog::adjustInputSize);

setMouseTracking(true);

}

protected:

void mousePressEvent(QMouseEvent\* event) override {

if (event->button() == Qt::LeftButton) {

dragPosition = event->globalPos() - frameGeometry().topLeft();

event->accept();

}

}

void mouseMoveEvent(QMouseEvent\* event) override {

if (event->buttons() & Qt::LeftButton) {

move(event->globalPos() - dragPosition);

event->accept();

}

}

void dragEnterEvent(QDragEnterEvent\* event) override {

if (event->mimeData()->hasText()) event->acceptProposedAction();

}

void dropEvent(QDropEvent\* event) override {

input->setText(input->toPlainText() + event->mimeData()->text());

event->acceptProposedAction();

}

private:

QTextEdit\* input;

QTextEdit\* output;

QPoint dragPosition;

void adjustInputSize() {

QString text = input->toPlainText();

int lines = text.split("\n").size();

int newHeight = std::min(std::max(100, lines \* 20 + 50), 1000);

input->setMinimumHeight(newHeight);

input->setMaximumHeight(newHeight);

}

void solveEquations() {

std::string expr = input->toPlainText().toStdString();

std::vector<std::string> equations;

std::stringstream ss(expr);

std::string line;

while (std::getline(ss, line)) {

if (!line.empty()) equations.push\_back(line);

}

QString result;

Qalculate calc;

py::scoped\_interpreter guard{};

py::module\_ sympy = py::module\_::import("sympy");

for (const auto& eq : equations) {

if (eq.find("d/d") != std::string::npos) {

// Derivative

std::string var = "x";

std::string func = eq.substr(eq.find("(") + 1, eq.find(")") - eq.find("(") - 1);

py::object x = sympy.attr("symbols")("x");

py::object expr = sympy.attr("sympify")(func);

py::object deriv = sympy.attr("diff")(expr, x);

result += QString("d/dx(%1) = %2\n").arg(QString::fromStdString(func), QString::fromStdString(deriv.attr("\_\_str\_\_")().cast<std::string>()));

} else if (eq.find("∫") != std::string::npos) {

// Integral

std::string bounds = eq.substr(eq.find("(") + 1, eq.find(")") - eq.find("(") - 1);

std::string func = eq.substr(eq.find(")") + 1, eq.find("dx") - eq.find(")") - 1);

auto [a, b] = parseBounds(bounds);

py::object x = sympy.attr("symbols")("x");

py::object expr = sympy.attr("sympify")(func);

py::object integral = sympy.attr("integrate")(expr, py::make\_tuple(x, a, b));

result += QString("∫(%1,%2) %3 dx = %4\n")

.arg(QString::number(a), QString::number(b), QString::fromStdString(func), QString::fromStdString(integral.attr("\_\_str\_\_")().cast<std::string>()));

} else if (eq.find("=") != std::string::npos) {

// Polynomial system

py::object x = sympy.attr("symbols")("x");

py::object y = sympy.attr("symbols")("y");

std::string eq\_clean = eq;

std::replace(eq\_clean.begin(), eq\_clean.end(), '=', '-');

equations.push\_back(eq\_clean);

if (equations.size() >= 2) {

py::object eq1 = sympy.attr("sympify")(equations[0]);

py::object eq2 = sympy.attr("sympify")(equations[1]);

py::object solutions = sympy.attr("solve")(py::make\_tuple(eq1, eq2), py::make\_tuple(x, y));

result += QString("System: %1, %2\nSolutions: %3\n")

.arg(QString::fromStdString(equations[0]), QString::fromStdString(equations[1]), QString::fromStdString(solutions.attr("\_\_str\_\_")().cast<std::string>()));

equations.clear();

}

} else {

result += QString("%1 = %2\n").arg(QString::fromStdString(eq), QString::fromStdString(calc.evaluate(eq)));

}

}

output->setText(result);

}

std::pair<double, double> parseBounds(const std::string& bounds) {

size\_t comma = bounds.find(",");

double a = std::stod(bounds.substr(0, comma));

double b = std::stod(bounds.substr(comma + 1));

return {a, b};

}

};

// Ramanujan Number Calculator

class RamanujanCalculatorDialog : public QDialog {

public:

RamanujanCalculatorDialog(QWidget\* parent) : QDialog(parent) {

setWindowFlags(Qt::Window | Qt::FramelessWindowHint);

setAcceptDrops(true);

QVBoxLayout\* layout = new QVBoxLayout(this);

input = new QTextEdit(this);

input->setPlaceholderText("Enter number theory functions (e.g., p(5), tau(7))");

input->setMinimumHeight(100);

input->setMaximumHeight(1000);

input->setAcceptDrops(true);

output = new QTextEdit(this);

output->setReadOnly(true);

QPushButton\* solveBtn = new QPushButton("Solve", this);

layout->addWidget(input);

layout->addWidget(solveBtn);

layout->addWidget(output);

connect(solveBtn, &QPushButton::clicked, this, &RamanujanCalculatorDialog::solveEquations);

connect(input, &QTextEdit::textChanged, this, &RamanujanCalculatorDialog::adjustInputSize);

setMouseTracking(true);

}

protected:

void mousePressEvent(QMouseEvent\* event) override {

if (event->button() == Qt::LeftButton) {

dragPosition = event->globalPos() - frameGeometry().topLeft();

event->accept();

}

}

void mouseMoveEvent(QMouseEvent\* event) override {

if (event->buttons() & Qt::LeftButton) {

move(event->globalPos() - dragPosition);

event->accept();

}

}

void dragEnterEvent(QDragEnterEvent\* event) override {

if (event->mimeData()->hasText()) event->acceptProposedAction();

}

void dropEvent(QDropEvent\* event) override {

input->setText(input->toPlainText() + event->mimeData()->text());

event->acceptProposedAction();

}

private:

QTextEdit\* input;

QTextEdit\* output;

QPoint dragPosition;

void adjustInputSize() {

QString text = input->toPlainText();

int lines = text.split("\n").size();

int newHeight = std::min(std::max(100, lines \* 20 + 50), 1000);

input->setMinimumHeight(newHeight);

input->setMaximumHeight(newHeight);

}

void solveEquations() {

std::string expr = input->toPlainText().toStdString();

std::vector<std::string> equations;

std::stringstream ss(expr);

std::string line;

while (std::getline(ss, line)) {

if (!line.empty()) equations.push\_back(line);

}

QString result;

py::scoped\_interpreter guard{};

py::module\_ sympy = py::module\_::import("sympy");

for (const auto& eq : equations) {

if (eq.find("p(") != std::string::npos) {

std::string n\_str = eq.substr(eq.find("(") + 1, eq.find(")") - eq.find("(") - 1);

int n = std::stoi(n\_str);

py::object partition = sympy.attr("partition")(n);

result += QString("p(%1) = %2 partitions\n").arg(n).arg(partition.cast<int>());

} else if (eq.find("tau(") != std::string::npos) {

std::string n\_str = eq.substr(eq.find("(") + 1, eq.find(")") - eq.find("(") - 1);

int n = std::stoi(n\_str);

py::object tau = sympy.attr("ramanujan\_tau")(n);

result += QString("tau(%1) = %2\n").arg(n).arg(tau.cast<long>());

} else {

result += QString("Invalid input: %1\n").arg(QString::fromStdString(eq));

}

}

output->setText(result);

}

};

// Calculus Button Field

class CalculusButtonField : public QDockWidget {

public:

CalculusButtonField(QWidget\* parent) : QDockWidget("Calculus Tools", parent) {

QWidget\* widget = new QWidget();

QVBoxLayout\* layout = new QVBoxLayout(widget);

QToolBar\* toolbar = new QToolBar(this);

input = new QTextEdit(this);

input->setPlaceholderText("Insert symbols (e.g., ∫, ∂, Σ)");

input->setMinimumHeight(100);

input->setMaximumHeight(1000);

input->setAcceptDrops(true);

toolbar->addAction("∫", [=]() { input->insertPlainText("∫(a,b) f(x) dx"); });

toolbar->addAction("∂", [=]() { input->insertPlainText("∂/∂x"); });

toolbar->addAction("Σ", [=]() { input->insertPlainText("Σ(n,a,b)"); });

toolbar->addAction("√", [=]() { input->insertPlainText("sqrt()"); });

toolbar->addAction("sin", [=]() { input->insertPlainText("sin()"); });

toolbar->addAction("cos", [=]() { input->insertPlainText("cos()"); });

toolbar->addAction("log", [=]() { input->insertPlainText("log()"); });

layout->addWidget(toolbar);

layout->addWidget(input);

setWidget(widget);

connect(input, &QTextEdit::textChanged, this, &CalculusButtonField::adjustInputSize);

}

protected:

void dragEnterEvent(QDragEnterEvent\* event) override {

if (event->mimeData()->hasText()) event->acceptProposedAction();

}

void dropEvent(QDropEvent\* event) override {

input->setText(input->toPlainText() + event->mimeData()->text());

event->acceptProposedAction();

}

private:

QTextEdit\* input;

void adjustInputSize() {

QString text = input->toPlainText();

int lines = text.split("\n").size();

int newHeight = std::min(std::max(100, lines \* 20 + 50), 1000);

input->setMinimumHeight(newHeight);

input->setMaximumHeight(newHeight);

}

};

// Detachable Browser Window

class BrowserWindow : public QMainWindow {

public:

BrowserWindow(const QString& title, QWidget\* parent = nullptr) : QMainWindow(parent) {

QWebEngineView\* view = new QWebEngineView(this);

QTextEdit\* summary = new QTextEdit(this);

summary->setReadOnly(true);

QVBoxLayout\* layout = new QVBoxLayout();

QWidget\* centralWidget = new QWidget();

layout->addWidget(view);

layout->addWidget(summary);

centralWidget->setLayout(layout);

setCentralWidget(centralWidget);

setWindowTitle(title);

views.push\_back(view);

summaries.push\_back(summary);

}

void setContent(const QString& html) {

views[0]->setHtml(html);

summaries[0]->setText(html);

}

private:

std::vector<QWebEngineView\*> views;

std::vector<QTextEdit\*> summaries;

};

// WebSocket callback

void on\_message(void\* user, const char\* data, size\_t len) {

std::string json\_data(data, len);

SearchResult result = {"wss://ligo.org/alerts", "Live Data", "Real-time event", 1.0, true};

results[0].push\_back(result);

sqlite3\_exec(db, ("INSERT INTO cache (url, title, summary, isLive) VALUES ('" + result.url + "', '" + result.title + "', '" + result.summary + "', 1)").c\_str(), nullptr, nullptr, nullptr);

}

// cURL callback

size\_t WriteCallback(void\* contents, size\_t size, size\_t nmemb, std::string\* data) {

data->append((char\*)contents, size \* nmemb);

return size \* nmemb;

}

// GPT-based summarization (Llama-3.1 fallback)

std::string SummarizeText(const std::string& text) {

py::scoped\_interpreter guard{};

py::module\_ transformers = py::module\_::import("transformers");

py::object summarizer = transformers.attr("pipeline")("summarization", "meta-llama/Llama-3.1-8B");

py::object summary = summarizer(text, py::arg("max\_length") = 100, py::arg("min\_length") = 30);

return summary[0].attr("summary\_text").cast<std::string>();

}

// OpenAI summarization with retry logic

std::string SummarizeWithOpenAI(const std::string& query) {

CURL\* curl = curl\_easy\_init();

std::string url = "https://api.openai.com/v1/chat/completions";

std::string response;

json payload = {

{"model", "gpt-4"},

{"messages", {{{"role", "user"}, {"content", "Summarize: " + query}}}},

{"max\_tokens", 100}

};

std::string data = payload.dump();

struct curl\_slist\* headers = nullptr;

headers = curl\_slist\_append(headers, "Content-Type: application/json");

headers = curl\_slist\_append(headers, ("Authorization: Bearer " + std::string(OPENAI\_API\_KEY)).c\_str());

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_POSTFIELDS, data.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_HTTPHEADER, headers);

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

int retries = 3;

while (retries--) {

CURLcode res = curl\_easy\_perform(curl);

long http\_code = 0;

curl\_easy\_getinfo(curl, CURLINFO\_RESPONSE\_CODE, &http\_code);

if (res == CURLE\_OK && http\_code == 200) {

json result = json::parse(response);

curl\_slist\_free\_all(headers);

curl\_easy\_cleanup(curl);

return result["choices"][0]["message"]["content"].get<std::string>();

} else if (http\_code == 429) {

std::this\_thread::sleep\_for(std::chrono::seconds(1 << (3 - retries)));

continue;

}

break;

}

curl\_slist\_free\_all(headers);

curl\_easy\_cleanup(curl);

return SummarizeText(query); // Fallback to Llama-3.1

}

// OAuth for cloud sync

std::string GetOAuthToken() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://<domain>.auth." + std::string(COGNITO\_REGION) + ".amazoncognito.com/oauth2/token";

std::string data = "grant\_type=client\_credentials&client\_id=" + std::string(COGNITO\_CLIENT\_ID) + "&client\_secret=your\_client\_secret";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_POSTFIELDS, data.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return "mock\_access\_token"; // Parse JSON for access\_token

}

// Cloud sync

void SyncCacheToCloud(const std::string& token) {

Aws::S3::Model::PutObjectRequest request;

request.SetBucket("coanqi-cache");

request.SetKey("cache.db");

request.SetCustomRequestHeader("Authorization", "Bearer " + token);

std::ifstream file("coanqi\_cache.db", std::ios::binary);

request.SetBody(std::make\_shared<Aws::Fstream>(file));

s3\_client->PutObject(request);

}

// Offline search

void OfflineSearch(const std::string& query, std::vector<SearchResult>& offlineResults) {

sqlite3\_stmt\* stmt;

sqlite3\_prepare\_v2(db, "SELECT url, title, summary, isLive FROM cache WHERE title LIKE ? OR summary LIKE ?", -1, &stmt, nullptr);

std::string pattern = "%" + query + "%";

sqlite3\_bind\_text(stmt, 1, pattern.c\_str(), -1, SQLITE\_STATIC);

sqlite3\_bind\_text(stmt, 2, pattern.c\_str(), -1, SQLITE\_STATIC);

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

SearchResult result;

result.url = (const char\*)sqlite3\_column\_text(stmt, 0);

result.title = (const char\*)sqlite3\_column\_text(stmt, 1);

result.summary = (const char\*)sqlite3\_column\_text(stmt, 2);

result.isLive = sqlite3\_column\_int(stmt, 3);

result.relevance = 0.9;

offlineResults.push\_back(result);

}

sqlite3\_finalize(stmt);

}

// Voice input

std::string ProcessVoiceInput() {

ps\_decoder\_t\* ps = ps\_init(cmd\_ln\_init(nullptr, ps\_args(), true, nullptr));

ps\_start\_utt(ps);

std::string text = "sample query"; // Replace with PocketSphinx

ps\_end\_utt(ps);

ps\_free(ps);

return text;

}

// Video input

std::string ProcessVideoInput() {

cv::VideoCapture cap(0);

cv::Mat frame;

cap >> frame;

std::string command = "submit query"; // Replace with OpenCV gesture recognition

cap.release();

return command;

}

// Visualization plugin

void RenderScatterPlot(QWidget\* parent, const std::vector<double>& x, const std::vector<double>& y) {

vtkSmartPointer<vtkScatterPlotMatrix> matrix = vtkSmartPointer<vtkScatterPlotMatrix>::New();

// Add x, y data (simplified)

}

// NASA API search (updated with new endpoints)

void SearchNASA(const std::string& query) {

CURL\* curl = curl\_easy\_init();

std::vector<std::string> endpoints = {

"https://api.nasa.gov/planetary/apod?api\_key=" + std::string(NASA\_API\_KEY\_1) + "&concept\_tags=True&keywords=" + query,

"https://api.nasa.gov/EPIC/api/natural?api\_key=" + std::string(NASA\_API\_KEY\_2),

"https://api.nasa.gov/DONKI/CMEAnalysis?api\_key=" + std::string(NASA\_API\_KEY\_2)

};

std::vector<std::string> titles = {"NASA APOD Result", "NASA EPIC Result", "NASA DONKI Result"};

std::vector<std::string> urls = {

"https://api.nasa.gov/planetary/apod",

"https://api.nasa.gov/EPIC/api/natural",

"https://api.nasa.gov/DONKI/CMEAnalysis"

};

for (size\_t i = 0; i < endpoints.size(); ++i) {

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, endpoints[i].c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

int retries = 3;

while (retries--) {

CURLcode res = curl\_easy\_perform(curl);

long http\_code = 0;

curl\_easy\_getinfo(curl, CURLINFO\_RESPONSE\_CODE, &http\_code);

if (res == CURLE\_OK && http\_code == 200) {

std::string summary = SummarizeWithOpenAI(response);

SearchResult result = {urls[i], titles[i], summary, 0.95, false};

results[i + 1].push\_back(result);

sqlite3\_exec(db, ("INSERT INTO cache (url, title, summary, isLive) VALUES ('" + result.url + "', '" + result.title + "', '" + result.summary + "', 0)").c\_str(), nullptr, nullptr, nullptr);

break;

} else if (http\_code == 429) {

std::this\_thread::sleep\_for(std::chrono::seconds(1 << (3 - retries)));

continue;

}

}

}

curl\_easy\_cleanup(curl);

}

// MAST API search

void SearchMAST(const std::string& query, std::vector<SearchResult>& mastResults) {

CURL\* curl = curl\_easy\_init();

std::string url = "https://mast.stsci.edu/api/v0.1/Download/file?uri=mast:HST/product/hst\_12345\_01\_acs\_f814w\_drz.fits&token=" + std::string(MAST\_API\_KEY);

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

if (res == CURLE\_OK) {

std::string summary = SummarizeWithOpenAI(response);

SearchResult result = {"https://mast.stsci.edu/api/v0.1/Download/file?uri=mast:HST/product/hst\_12345\_01\_acs\_f814w\_drz.fits", "MAST HST Infrared", summary, 0.95, false};

mastResults.push\_back(result);

sqlite3\_exec(db, ("INSERT INTO cache (url, title, summary, isLive) VALUES ('" + result.url + "', '" + result.title + "', '" + result.summary + "', 0)").c\_str(), nullptr, nullptr, nullptr);

}

curl\_easy\_cleanup(curl);

}

// JPL Horizons API

std::string FetchHorizons() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd.jpl.nasa.gov/api/horizons.api?format=text&COMMAND='499'&OBJ\_DATA='YES'&MAKE\_EPHEM='YES'&EPHEM\_TYPE='OBSERVER'&CENTER='500@399'&START\_TIME='2006-01-01'&STOP\_TIME='2006-01-20'&STEP\_SIZE='1%20d'&QUANTITIES='1,9,20,23,24,29'";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL JD-Cal JD

std::string FetchJDCalJD() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/jd\_cal.api?jd=2451544&format=s";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL JD-Cal CD

std::string FetchJDCalCD() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/jd\_cal.api?cd=2000-01-01\_12:00";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL Periodic Orbits Earth-Moon

std::string FetchPeriodicEarthMoon() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=earth-moon&family=halo&libr=1&branch=N";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL Periodic Orbits Jupiter-Europa

std::string FetchPeriodicJupiterEuropa() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=jupiter-europa&family=dro";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL Periodic Orbits Saturn-Enceladus

std::string FetchPeriodicSaturnEnceladus() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=saturn-enceladus&family=vertical&libr=2&periodmax=1&periodunits=d";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL Periodic Orbits Saturn-Titan

std::string FetchPeriodicSaturnTitan() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=saturn-titan&family=butterfly&jacobimin=3&stabmax=1&branch=N";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// JPL Periodic Orbits Mars-Phobos

std::string FetchPeriodicMarsPhobos() {

CURL\* curl = curl\_easy\_init();

std::string url = "https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=mars-phobos&family=resonant&jacobimin=3&stabmax=1&branch=21";

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

curl\_easy\_cleanup(curl);

return response;

}

// Search function (updated with JPL APIs)

void PerformSearch(const std::string& query, std::vector<std::string>& focus, bool online, const std::string& oauth\_token) {

if (!online) {

std::vector<SearchResult> offlineResults;

OfflineSearch(query, offlineResults);

for (int i = 0; i < MAX\_WINDOWS && i < offlineResults.size(); ++i) {

results[i].push\_back(offlineResults[i]);

}

return;

}

std::vector<SearchResult> nasaResults, mastResults;

if (std::find(focus.begin(), focus.end(), "NASA") != focus.end()) {

SearchNASA(query, nasaResults);

results[1] = nasaResults;

}

if (std::find(focus.begin(), focus.end(), "STScI") != focus.end() ||

std::find(focus.begin(), focus.end(), "Hubble") != focus.end() ||

std::find(focus.begin(), focus.end(), "ACS Hubble Ultra Deep Field") != focus.end()) {

SearchMAST(query, mastResults);

results[2] = mastResults;

}

// Preload specified links with OpenAI summaries

results[3].push\_back({"https://mast.stsci.edu/api/v0.1/Download/file?uri=mast:HST/product/hst\_12345\_01\_acs\_f814w\_drz.fits", "MAST ACS F814W Infrared", SummarizeWithOpenAI("Hubble infrared data"), 0.95, false});

results[4].push\_back({"wss://eventhorizontelescope.org/data", "EHT Live Infrared Data", SummarizeWithOpenAI("Real-time EHT data"), 1.0, true});

results[5].push\_back({"https://apod.nasa.gov/apod/image/2507/m31\_infrared.jpg", "NASA M31 Infrared", SummarizeWithOpenAI("Andromeda infrared image"), 0.95, false});

results[6].push\_back({"wss://ligo.org/alerts", "LIGO GW Infrared Correlations", SummarizeWithOpenAI("Real-time GW alerts"), 1.0, true});

// JPL APIs

if (query.find("ephemeris") != std::string::npos || query.find("horizons") != std::string::npos) {

std::string horizons = FetchHorizons();

results[7].push\_back({"https://ssd.jpl.nasa.gov/api/horizons.api", "JPL Horizons Ephemeris", SummarizeWithOpenAI(horizons), 0.95, false});

}

if (query.find("jd to date") != std::string::npos) {

std::string jdcal = FetchJDCalJD();

results[8].push\_back({"https://ssd-api.jpl.nasa.gov/jd\_cal.api?jd=2451544&format=s", "JPL JD-Cal JD to Date", SummarizeWithOpenAI(jdcal), 0.95, false});

}

if (query.find("date to jd") != std::string::npos) {

std::string jdcal = FetchJDCalCD();

results[9].push\_back({"https://ssd-api.jpl.nasa.gov/jd\_cal.api?cd=2000-01-01\_12:00", "JPL JD-Cal Date to JD", SummarizeWithOpenAI(jdcal), 0.95, false});

}

if (query.find("earth-moon halo") != std::string::npos) {

std::string orbits = FetchPeriodicEarthMoon();

results[10].push\_back({"https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=earth-moon&family=halo&libr=1&branch=N", "JPL Periodic Orbits Earth-Moon", SummarizeWithOpenAI(orbits), 0.95, false});

}

if (query.find("jupiter-europa dro") != std::string::npos) {

std::string orbits = FetchPeriodicJupiterEuropa();

results[11].push\_back({"https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=jupiter-europa&family=dro", "JPL Periodic Orbits Jupiter-Europa", SummarizeWithOpenAI(orbits), 0.95, false});

}

if (query.find("saturn-enceladus vertical") != std::string::npos) {

std::string orbits = FetchPeriodicSaturnEnceladus();

results[12].push\_back({"https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=saturn-enceladus&family=vertical&libr=2&periodmax=1&periodunits=d", "JPL Periodic Orbits Saturn-Enceladus", SummarizeWithOpenAI(orbits), 0.95, false});

}

if (query.find("saturn-titan butterfly") != std::string::npos) {

std::string orbits = FetchPeriodicSaturnTitan();

results[13].push\_back({"https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=saturn-titan&family=butterfly&jacobimin=3&stabmax=1&branch=N", "JPL Periodic Orbits Saturn-Titan", SummarizeWithOpenAI(orbits), 0.95, false});

}

if (query.find("mars-phobos resonant") != std::string::npos) {

std::string orbits = FetchPeriodicMarsPhobos();

results[14].push\_back({"https://ssd-api.jpl.nasa.gov/periodic\_orbits.api?sys=mars-phobos&family=resonant&jacobimin=3&stabmax=1&branch=21", "JPL Periodic Orbits Mars-Phobos", SummarizeWithOpenAI(orbits), 0.95, false});

}

struct lws\_context\* ws\_context = lws\_create\_context(nullptr);

lws\_connect(ws\_context, "eventhorizontelescope.org", 443, "/data", on\_message, nullptr);

lws\_connect(ws\_context, "skaobservatory.org", 443, "/realtime", on\_message, nullptr);

lws\_connect(ws\_context, "ligo.org", 443, "/alerts", on\_message, nullptr);

lws\_connect(ws\_context, "fast.bao.ac.cn", 443, "/realtime", on\_message, nullptr);

CURL\* curl = curl\_easy\_init();

for (int i = 15; i < MAX\_WINDOWS && i < focus.size(); ++i) {

std::string url = "https://api.example.com/search?q=" + query + "&source=" + focus[i];

std::string response;

curl\_easy\_setopt(curl, CURLOPT\_URL, url.c\_str());

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, WriteCallback);

curl\_easy\_setopt(curl, CURLOPT\_WRITEDATA, &response);

CURLcode res = curl\_easy\_perform(curl);

if (res == CURLE\_OK) {

std::string summary = SummarizeWithOpenAI(response);

SearchResult result = {"https://example.com", "Sample Result", summary, 0.95, false};

results[i].push\_back(result);

sqlite3\_exec(db, ("INSERT INTO cache (url, title, summary, isLive) VALUES ('" + result.url + "', '" + result.title + "', '" + result.summary + "', 0)").c\_str(), nullptr, nullptr, nullptr);

}

}

curl\_easy\_cleanup(curl);

lws\_context\_destroy(ws\_context);

SyncCacheToCloud(oauth\_token);

}

// Qt Main Window

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

MainWindow() {

// Win32: System tray

#ifdef \_WIN32

NOTIFYICONDATA nid = {sizeof(nid)};

nid.hWnd = (HWND)winId();

nid.uID = 1;

nid.uFlags = NIF\_ICON | NIF\_TIP;

nid.hIcon = LoadIcon(GetModuleHandle(nullptr), "Z.ico");

strcpy(nid.szTip, "CoAnQi");

Shell\_NotifyIcon(NIM\_ADD, &nid);

#endif

QWidget\* centralWidget = new QWidget(this);

QVBoxLayout\* layout = new QVBoxLayout(centralWidget);

// Top bar (Firefox-like)

QHBoxLayout\* topBar = new QHBoxLayout();

QPushButton\* backBtn = new QPushButton("Back", this);

QPushButton\* forwardBtn = new QPushButton("Forward", this);

QPushButton\* refreshBtn = new QPushButton("Refresh", this);

QLineEdit\* queryField = new QLineEdit(this);

queryField->setMaxLength(MAX\_QUERY\_LENGTH);

queryField->setPlaceholderText("Search high-energy datasets...");

QPushButton\* voiceBtn = new QPushButton("🎤", this);

QPushButton\* videoBtn = new QPushButton("📷", this);

QPushButton\* sciCalcBtn = new QPushButton("🧮", this);

QPushButton\* ramCalcBtn = new QPushButton("🧮R", this);

QPushButton\* calcBtnField = new QPushButton("🧮C", this);

QLabel\* logo = new QLabel("<b>CoAnQi (Cosmic Analysis and Quantum Intelligence)</b>", this);

logo->setStyleSheet("font-size: 24px; color: #2a5298;");

QPushButton\* menuBtn = new QPushButton("☰", this);

topBar->addWidget(backBtn);

topBar->addWidget(forwardBtn);

topBar->addWidget(refreshBtn);

topBar->addWidget(queryField);

topBar->addWidget(voiceBtn);

topBar->addWidget(videoBtn);

topBar->addWidget(sciCalcBtn);

topBar->addWidget(ramCalcBtn);

topBar->addWidget(calcBtnField);

topBar->addWidget(logo);

topBar->addWidget(menuBtn);

layout->addLayout(topBar);

// Focus list

QTextEdit\* focusField = new QTextEdit(this);

QString focusText;

for (const auto& item : focusList) focusText += QString::fromStdString(item) + "\n";

focusField->setText(focusText);

layout->addWidget(focusField);

// Tabbed browser windows

QTabWidget\* tabs = new QTabWidget(this);

tabs->setTabsClosable(true);

tabs->setMovable(true);

browserWindows = new BrowserWindow\*[MAX\_WINDOWS];

for (int i = 0; i < MAX\_WINDOWS; ++i) {

browserWindows[i] = new BrowserWindow(QString("Tab %1").arg(i + 1), this);

tabs->addTab(new QWidget(), QString("Tab %1").arg(i + 1));

}

layout->addWidget(tabs);

// Visualization sidebar

QDockWidget\* sidebar = new QDockWidget("Visualizations", this);

QWidget\* visWidget = new QWidget();

QVBoxLayout\* visLayout = new QVBoxLayout(visWidget);

visLayout->addWidget(new QLabel("Dataset Graph Placeholder"));

sidebar->setWidget(visWidget);

addDockWidget(Qt::LeftDockWidgetArea, sidebar);

// Calculus button field

CalculusButtonField\* calcField = new CalculusButtonField(this);

addDockWidget(Qt::RightDockWidgetArea, calcField);

// Calculators

ScientificCalculatorDialog\* sciCalcDialog = new ScientificCalculatorDialog(this);

sciCalcDialog->move(50, 50);

sciCalcDialog->show();

RamanujanCalculatorDialog\* ramCalcDialog = new RamanujanCalculatorDialog(this);

ramCalcDialog->move(100, 100);

ramCalcDialog->show();

setCentralWidget(centralWidget);

// Initialize SQLite and AWS

sqlite3\_open("coanqi\_cache.db", &db);

sqlite3\_exec(db, "CREATE TABLE IF NOT EXISTS cache (url TEXT, title TEXT, summary TEXT, isLive INTEGER)", nullptr, nullptr, nullptr);

Aws::SDKOptions options;

Aws::InitAPI(options);

s3\_client = new Aws::S3::S3Client();

cognito\_client = new Aws::CognitoIdentityProvider::CognitoIdentityProviderClient();

// OAuth login

std::string oauth\_token = GetOAuthToken();

// Connect signals

connect(queryField, &QLineEdit::returnPressed, [=]() {

std::string query = queryField->text().toStdString();

if (query.length() > MAX\_QUERY\_LENGTH) {

QMessageBox::warning(this, "Error", "Query exceeds 3000 characters!");

return;

}

bool online = true; // Check connectivity

PerformSearch(query, focusList, online, oauth\_token);

for (int i = 0; i < MAX\_WINDOWS; ++i) {

QString html = "<ul>";

for (const auto& result : results[i]) {

QString live = result.isLive ? " [Live]" : "";

html += QString("<li><a href='%1'>%2</a>%3: %4</li>")

.arg(QString::fromStdString(result.url))

.arg(QString::fromStdString(result.title))

.arg(live)

.arg(QString::fromStdString(result.summary));

}

html += "</ul>";

browserWindows[i]->setContent(html);

}

});

connect(tabs, &QTabWidget::tabBarDoubleClicked, [=](int index) {

BrowserWindow\* window = browserWindows[index];

window->show();

tabs->removeTab(index);

});

connect(voiceBtn, &QPushButton::clicked, [=]() {

queryField->setText(QString::fromStdString(ProcessVoiceInput()));

});

connect(videoBtn, &QPushButton::clicked, [=]() {

if (ProcessVideoInput() == "submit query") {

QKeyEvent\* event = new QKeyEvent(QEvent::KeyPress, Qt::Key\_Return, Qt::NoModifier);

QCoreApplication::postEvent(queryField, event);

}

});

connect(sciCalcBtn, &QPushButton::clicked, [=]() {

sciCalcDialog->show();

});

connect(ramCalcBtn, &QPushButton::clicked, [=]() {

ramCalcDialog->show();

});

connect(calcBtnField, &QPushButton::clicked, [=]() {

calcField->show();

});

connect(focusField, &QTextEdit::textChanged, [=]() {

focusList.clear();

QStringList lines = focusField->toPlainText().split("\n");

for (const auto& line : lines) {

if (!line.isEmpty()) focusList.push\_back(line.toStdString());

}

});

}

~MainWindow() {

for (int i = 0; i < MAX\_WINDOWS; ++i) delete browserWindows[i];

delete[] browserWindows;

sqlite3\_close(db);

delete s3\_client;

delete cognito\_client;

Aws::ShutdownAPI(Aws::SDKOptions());

#ifdef \_WIN32

NOTIFYICONDATA nid = {sizeof(nid)};

nid.uID = 1;

Shell\_NotifyIcon(NIM\_DELETE, &nid);

#endif

}

private:

BrowserWindow\*\* browserWindows;

};

// Main function

int main(int argc, char\* argv[]) {

QApplication app(argc, argv);

MainWindow window;

window.setWindowTitle("CoAnQi");

window.setWindowIcon(QIcon("Z.png"));

window.show();

return app.exec();

}