Maria Arcocha Morales #636952
Diego Ponce González #594623
Alí Cano García Escobar #597058
Karol Castillo Martínez #642020

Script

```
# -- coding: utf-8 --
import gate_api
from gate_api.exceptions import ApiException, GateApiException
import pandas as pd
import numpy as np
import time
import sys
import logging
from datetime import datetime
from twilio.rest import Client
import joblib
import pandas_ta as ta
import os
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
import csv

# --- Configuración de logging ---
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')
logging.getLogger("urllib3").setLevel(logging.WARNING)
logging.getLogger("tensorflow").setLevel(logging.ERROR)

#
==========================================================================
# 1) PARÁMETROS DE USUARIO
#
==========================================================================
# 🚨 ADVERTENCIA: Claves expuestas.
API_KEY = "91164d7defcdbd2e0e7eb130c31abf33"
API_SECRET =
"06607d6cea3d69efae6e80e22d4af397ba4f12e9f70d1bd2dd3c5a815eecf19f"

# --- Parámetros del Contrato ---
SYMBOL = "BTC_USDT"
SETTLE = "usdt"
TIMEFRAME = "1m"
LEVERAGE = "10"
```

```
# --- Parámetros de Estrategia (ACTUALIZADO) ---
RSI_PERIOD = 14
RSI_BUY_LEVEL = 30
RSI_SELL_LEVEL = 70
RSI_EXIT_TARGET = 50
VOL_PERIOD = 12
VOL_LOW_MAX = 1.1
VOL_HIGH_MAX = 1.13
ATR_SL_MULTIPLIER = 2.5 # <--- MODIFICADO: Stop Loss fijo basado en ATR a 2.5
TRAILING_SL_START_PERCENT = 0.005 # NUEVO: 0.5% de ganancia para activar
Trailing Stop

# --- Parámetros LSTM ---
N_LAGS = 60
MODEL_FILE = 'lstm_model_btc_1m.h5'
SCALER_FILE = 'scaler_btc_1m.pkl'
BASE_QTY = 0.010
MODIFIER_HIGH_CONFIDENCE = 1.5
MODIFIER_LOW_CONFIDENCE = 0.5
LSTM_CONFIRMATION_THRESHOLD = 0.0005

# --- Entrenamiento ---
TRAIN_DATA_LIMIT = 2000
TRAIN_EPOCHS = 20
TRAIN_BATCH_SIZE = 32

# --- Logs ---
LOG_TRADES_FILE = 'trade_history_btc_1m.csv'
LOG_PREDICTIONS_FILE = 'model_predictions_btc_1m.csv'

# --- Twilio ---
account_sid = 'AC64954a2119e4de5caf95aba1cef9ce47'
auth_token = 'f1a5fd55ae0777ff9213ac26f2cf3c50'
twilio_client = Client(account_sid, auth_token)
whatsapp_from = 'whatsapp:+14155238886'
whatsapp_to = 'whatsapp:+5218121487240'

#
================================================================================
# 2) FUNCIONES AUXILIARES
#
================================================================================
def setup_log_files():
    if not os.path.exists(LOG_PREDICTIONS_FILE):
        with open(LOG_PREDICTIONS_FILE, mode='w', newline='') as f:
            writer = csv.writer(f)
            writer.writerow(["Timestamp", "Precio_Cierre", "Prediccion_LSTM", "RSI", "ATR"])
    if not os.path.exists(LOG_TRADES_FILE):
```

```
        with open(LOG_TRADES_FILE, mode='w', newline='') as f:
            writer = csv.writer(f)
            # Columna 'SL_Inicial' ahora almacena el nivel de precio del SL inicial/final
            writer.writerow(["Timestamp_Cierre", "Tipo", "Precio_Entrada", "RSI_Entrada",
"ATR_Entrada", "SL_Inicial", "Precio_Salida", "Motivo_Cierre", "PNL_Puntos"])

def log_prediction(timestamp, price, prediction, rsi, atr):
    try:
        with open(LOG_PREDICTIONS_FILE, mode='a', newline='') as f:
            writer = csv.writer(f)
            writer.writerow([timestamp, f"{price:.4f}", f"{prediction:.4f}", f"{rsi:.2f}", f"{atr:.4f}"])
    except Exception as e:
        logging.error(f"Error log predicciones: {e}")

def log_trade(timestamp, trade_type, entry_price, entry_rsi, entry_atr, initial_sl, exit_price,
reason, pnl):
    try:
        with open(LOG_TRADES_FILE, mode='a', newline='') as f:
            writer = csv.writer(f)
            # La columna SL_Inicial ahora acepta el nivel de precio del SL
            writer.writerow([timestamp, trade_type, f"{entry_price:.4f}", f"{entry_rsi:.2f}",
f"{entry_atr:.4f}", f"{initial_sl:.4f}", f"{exit_price:.4f}", reason, f"{pnl:.4f}"])
    except Exception as e:
        logging.error(f"Error log trades: {e}")

def send_whatsapp_message(message):
    try:
        msg = twilio_client.messages.create(body=message, from_=whatsapp_from,
to=whatsapp_to)
        print("Notificación enviada a WhatsApp, SID:", msg.sid)
    except Exception as e:
        logging.error("Error al enviar notificación WhatsApp: %s", str(e))

def init_exchange():
    is_testnet = True
    host_url = "https://api-testnet.gateapi.io/api/v4"
    logging.info("[INIT] Conectando al Testnet de Gate.io FUTURES...")
    configuration = gate_api.Configuration(key=API_KEY, secret=API_SECRET,
host=host_url)
    api_client = gate_api.ApiClient(configuration)
    futures_api = gate_api.FuturesApi(api_client)
    try:
        futures_api.list_futures_accounts(settle=SETTLE)
        logging.info("[OK] Credenciales de API válidas.")
        try:
            futures_api.update_position_leverage(settle=SETTLE, contract=SYMBOL,
leverage=LEVERAGE)
            logging.info(f"[INIT] Apalancamiento para {SYMBOL} ajustado a {LEVERAGE}x")
```

```
        except GateApiException as ex:
            logging.warning(f"[WARN] No se pudo ajustar leverage: {ex.message}")
        return futures_api
    except GateApiException as ex:
        logging.critical(f"[ERROR FATAL] Fallo conexión API: {ex.message}")
        sys.exit(1)

def cancel_all_open_orders(client):
    try:
        logging.info(f"🧹 Limpiando órdenes pendientes para {SYMBOL}...")
        client.cancel_futures_orders(SETTLE, contract=SYMBOL)
        logging.info("✅ Todas las órdenes pendientes han sido canceladas.")
    except Exception as e:
        logging.error(f"Error intentando limpiar órdenes: {e}")

def get_contract_details(client):
    try:
        contract = client.get_futures_contract(settle=SETTLE, contract=SYMBOL)
        return {"quanto_multiplier": float(contract.quanto_multiplier)}
    except Exception as ex:
        logging.critical(f"Error contrato: {ex}")
        return None

def place_market_order(client, size):
    try:
        order = gate_api.FuturesOrder(contract=SYMBOL, size=int(size), price='0', tif='ioc')
        created_order = client.create_futures_order(SETTLE, futures_order=order)
        time.sleep(1)
        filled_order = client.get_futures_order(SETTLE, created_order.id)
        if filled_order.status == 'finished' and filled_order.fill_price:
            logging.info(f"✅ Orden ejecutada: ${float(filled_order.fill_price):,.4f}")
            return float(filled_order.fill_price)
        else:
            logging.error(f"Orden no ejecutada correctamente: {filled_order.status}")
            return None
    except GateApiException as ex:
        logging.error(f"Error API orden: {ex.message}")
        return None

def get_current_position(client):
    try:
        position = client.get_position(SETTLE, SYMBOL)
        return {"size": int(position.size), "entry_price": float(position.entry_price)}
    except Exception:
        return {"size": 0, "entry_price": 0.0}

def fetch_data_and_indicators(client):
    try:
```

```
        bars = client.list_futures_candlesticks(settle=SETTLE, contract=SYMBOL,
interval=TIMEFRAME, limit=200)
        df = pd.DataFrame([{"timestamp": int(bar.t), "close": float(bar.c), "high": float(bar.h),
"low": float(bar.l)} for bar in bars])
        df["dt"] = pd.to_datetime(df["timestamp"], unit='s')
        df = df.sort_values(by="timestamp").reset_index(drop=True)
        df[f'ATR_14'] = df.ta.atr(length=14)
        df[f'RSI_{RSI_PERIOD}'] = df.ta.rsi(length=RSI_PERIOD)
        return df
    except Exception as e:
        logging.error(f"Error indicadores: {e}")
        return None

def get_lstm_prediction(df_datos, modelo, scaler, n_lags):
    df_clean = df_datos.iloc[:-1].dropna(subset=['close'])
    if len(df_clean) < n_lags: return None
    try:
        ultimos_datos = df_clean['close'].tail(n_lags).values.reshape(-1, 1)
        datos_normalizados = scaler.transform(ultimos_datos)
        datos_para_predecir = np.reshape(datos_normalizados, (1, n_lags, 1))
        prediccion = scaler.inverse_transform(modelo.predict(datos_para_predecir,
verbose=0))
        return prediccion[0][0]
    except Exception: return None

#
==========================================================================
# 3) ENTRENAMIENTO
#
==========================================================================
def fetch_training_data(client, limit=2000):
    logging.info(f"Descargando {limit} velas para entrenamiento...")
    try:
        bars = client.list_futures_candlesticks(settle=SETTLE, contract=SYMBOL,
interval=TIMEFRAME, limit=limit)
        df = pd.DataFrame([{"timestamp": int(bar.t), "close": float(bar.c)} for bar in bars])
        return
df.sort_values(by="timestamp").reset_index(drop=True)['close'].values.reshape(-1, 1)
    except Exception as e:
        logging.error(f"Error datos entrenamiento: {e}")
        return None

def train_and_save_model(client, scaler_path, model_path):
    logging.warning(f"Entrenando modelo LSTM para {SYMBOL} ({TIMEFRAME})...")
    send_whatsapp_message(f"🤖 Entrenando modelo LSTM para {SYMBOL}
({TIMEFRAME})...")
    data = fetch_training_data(client, limit=TRAIN_DATA_LIMIT)
    if data is None or len(data) < N_LAGS + 50: sys.exit(1)
```

```
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data)
    joblib.dump(scaler, scaler_path)

    X_train, y_train = [], []
    for i in range(N_LAGS, len(scaled_data)):
        X_train.append(scaled_data[i-N_LAGS:i, 0])
        y_train.append(scaled_data[i, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))

    model = Sequential()
    model.add(LSTM(units=50, return_sequences=True, input_shape=(N_LAGS, 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=50, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(units=25))
    model.add(Dense(units=1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(X_train, y_train, epochs=TRAIN_EPOCHS, batch_size=TRAIN_BATCH_SIZE,
verbose=1)
    model.save(model_path)
    send_whatsapp_message(f"✅ Modelo LSTM ({SYMBOL} {TIMEFRAME}) guardado.")

#
==========================================================================
# 4) MAIN (BTC 1m) - RSI 50 TARGET & ATR SL & TRAILING STOP
#
==========================================================================

def main():
    client = init_exchange()
    setup_log_files()
    cancel_all_open_orders(client)

    if not os.path.exists(MODEL_FILE) or not os.path.exists(SCALER_FILE):
        train_and_save_model(client, SCALER_FILE, MODEL_FILE)

    try:
        modelo_lstm = tf.keras.models.load_model(MODEL_FILE)
        scaler_lstm = joblib.load(SCALER_FILE)
        logging.info("[OK] Modelo y Scaler cargados.")
    except Exception as e:
        logging.critical(f"Error cargando modelo: {e}")
        sys.exit(1)

    # Variables de Estado (ACTUALIZADO)
```

```
    position_type = None
    position_size_contracts = 0
    entry_price = 0.0
    entry_sl_level = 0.0 # Nivel de precio del Stop Loss

    entry_rsi = 0.0
    entry_atr = 0.0
    last_known_rsi = None
    trailing_activated = False # ESTADO DEL TRAILING STOP

    contract_details = get_contract_details(client)
    if not contract_details: sys.exit(1)
    contract_multiplier = contract_details["quanto_multiplier"]

    logging.info(f"✅ Bot Iniciado: {SYMBOL} ({TIMEFRAME}) [MODO: RSI 50 SALIDA + ATR SL + Trailing Stop]")
    send_whatsapp_message(f"✅ Bot Iniciado: {SYMBOL} ({TIMEFRAME}) [MODO: RSI 50 SALIDA + ATR SL + Trailing Stop]")

    while True:
        try:
            current_time = time.time()
            current_timestamp_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            # 1. Sincronización
            actual_position = get_current_position(client)
            actual_position_size = actual_position.get('size', 0)

            # Sincronización de posición cerrada externamente (reseteo de variables de estado)
            if actual_position_size == 0 and position_type is not None:
                logging.warning("¡Desincronización! Posición cerrada externamente.")
                log_trade(current_timestamp_str, position_type, entry_price, entry_rsi, entry_atr, entry_sl_level, 0.0, "Desync/Manual", 0.0)
                position_type, position_size_contracts, entry_sl_level = None, 0, 0.0
                entry_price = 0.0
                trailing_activated = False

            # 2. Obtener Datos
            df = fetch_data_and_indicators(client)
            if df is None or len(df) < max(VOL_PERIOD, N_LAGS, RSI_PERIOD):
                logging.warning("Datos insuficientes, reintentando...")
                time.sleep(10)
                continue

            # 3. Calcular Volatilidad
            df['low_pct_change'] = df['low'].pct_change() * 100
            df['high_pct_change'] = df['high'].pct_change() * 100
            df['vol_low_pct'] = df['low_pct_change'].rolling(VOL_PERIOD).std()
```

```
            df['vol_high_pct'] = df['high_pct_change'].rolling(VOL_PERIOD).std()

            df_clean = df.dropna().reset_index(drop=True)
            if len(df_clean) < 2:
                time.sleep(10)
                continue

            # --- REFERENCIAS DE VELA ---
            signal_row = df_clean.iloc[-2] # Vela Cerrada (Entradas)
            current_row = df_clean.iloc[-1] # Vela Actual (Salidas/Cierre)

            current_price = current_row["close"]

            atr_signal = signal_row[f'ATR_14']
            rsi_signal = signal_row[f'RSI_{RSI_PERIOD}']
            rsi_live = current_row[f'RSI_{RSI_PERIOD}'] # RSI en la vela actual para salidas

            valor_volatilidad_baja = signal_row['vol_low_pct']
            valor_volatilidad_alta = signal_row['vol_high_pct']
            permitir_compra = (valor_volatilidad_baja <= VOL_LOW_MAX)
            permitir_venta = (valor_volatilidad_alta <= VOL_HIGH_MAX)

            # Warm-up y Sincronización RSI
            if last_known_rsi is None:
                logging.info(f"🔥 Calibrando RSI con vela cerrada ({rsi_signal:.2f})...")
                last_known_rsi = rsi_signal
                sleep_time = 60 - (time.time() % 60)
                time.sleep(sleep_time + 2)
                continue

            previous_rsi_signal = last_known_rsi

            # --- 4A. GESTIÓN DEL TRAILING STOP (NUEVO) ---
            if position_type and position_type in ('LONG', 'SHORT'):
                price_change_pct = (current_price - entry_price) / entry_price
                sl_buffer = entry_atr * ATR_SL_MULTIPLIER

                if position_type == 'LONG':
                    current_pnl_pct = price_change_pct

                    # 1. Activación del Trailing Stop
                    if current_pnl_pct >= TRAILING_SL_START_PERCENT and not
trailing_activated:
                        trailing_activated = True
                        logging.info(f"🟢 Trailing Stop ACTIVADO. Ganancia >
{TRAILING_SL_START_PERCENT * 100:.2f}%.")

                    # 2. Movimiento del Trailing Stop
```

```
                if trailing_activated:
                    new_sl = current_price - sl_buffer
                    # Solo movemos el SL si el nuevo nivel es MÁS ALTO (protegiendo más
ganancia)
                    if new_sl > entry_sl_level:
                        entry_sl_level = new_sl
                        logging.info(f"⬆️ Trailing SL movido a: {entry_sl_level:,.4f}")

            elif position_type == 'SHORT':
                current_pnl_pct = -price_change_pct # PNL es inverso para SHORT

                # 1. Activación del Trailing Stop
                if current_pnl_pct >= TRAILING_SL_START_PERCENT and not
trailing_activated:
                    trailing_activated = True
                    logging.info(f"🟣 Trailing Stop ACTIVADO. Ganancia >
{TRAILING_SL_START_PERCENT * 100:.2f}%.")

                # 2. Movimiento del Trailing Stop
                if trailing_activated:
                    new_sl = current_price + sl_buffer
                    # Solo movemos el SL si el nuevo nivel es MÁS BAJO (protegiendo más
ganancia)
                    if new_sl < entry_sl_level:
                        entry_sl_level = new_sl
                        logging.info(f"⬇️ Trailing SL movido a: {entry_sl_level:,.4f}")


        # --- 4B. VERIFICACIÓN DE SALIDA (SL/TP) ---
        if position_type == 'LONG':
            # Stop Loss ATR (Ahora incluye Trailing SL)
            if current_price <= entry_sl_level:
                reason = "SL ATR/Trailing"
                logging.critical(f"🛑 STOP LOSS LONG activado! Precio: {current_price:,.2f} vs
SL: {entry_sl_level:,.2f}")
                fill_close = place_market_order(client, -position_size_contracts)
                if fill_close:
                    pnl = fill_close - entry_price
                    log_trade(current_timestamp_str, position_type, entry_price, entry_rsi,
entry_atr, entry_sl_level, fill_close, reason, pnl)
                    send_whatsapp_message(f"🛑 SL LONG ({reason}). PNL: {pnl:,.2f}. Salida
@ {fill_close:,.2f}")
                    position_type, position_size_contracts, entry_sl_level = None, 0, 0.0
                    entry_price = 0.0
                    trailing_activated = False
                    last_known_rsi = rsi_signal
                    continue
```

```
            # Take Profit RSI 50
            elif rsi_live >= RSI_EXIT_TARGET:
                logging.info(f"🎯 RSI tocó 50 ({rsi_live:.2f}). Cerrando LONG.")
                fill_close = place_market_order(client, -position_size_contracts)
                if fill_close:
                    pnl = fill_close - entry_price
                    log_trade(current_timestamp_str, position_type, entry_price, entry_rsi,
entry_atr, entry_sl_level, fill_close, "RSI 50 Target", pnl)
                    send_whatsapp_message(f"🎯 SALIDA RSI 50. PNL: {pnl:,.2f}")
                    position_type, position_size_contracts, entry_sl_level = None, 0, 0.0
                    entry_price = 0.0
                    trailing_activated = False
                    last_known_rsi = rsi_signal
                    continue

        elif position_type == 'SHORT':
            # Stop Loss ATR (Ahora incluye Trailing SL)
            if current_price >= entry_sl_level:
                reason = "SL ATR/Trailing"
                logging.critical(f"🛑 STOP LOSS SHORT activado! Precio: {current_price:,.2f}
vs SL: {entry_sl_level:,.2f}")
                fill_close = place_market_order(client, -position_size_contracts)
                if fill_close:
                    pnl = entry_price - fill_close
                    log_trade(current_timestamp_str, position_type, entry_price, entry_rsi,
entry_atr, entry_sl_level, fill_close, reason, pnl)
                    send_whatsapp_message(f"🛑 SL SHORT ({reason}). PNL: {pnl:,.2f}. Salida
@ {fill_close:,.2f}")
                    position_type, position_size_contracts, entry_sl_level = None, 0, 0.0
                    entry_price = 0.0
                    trailing_activated = False
                    last_known_rsi = rsi_signal
                    continue

            # Take Profit RSI 50
            elif rsi_live <= RSI_EXIT_TARGET:
                logging.info(f"🎯 RSI tocó 50 ({rsi_live:.2f}). Cerrando SHORT.")
                fill_close = place_market_order(client, -position_size_contracts)
                if fill_close:
                    pnl = entry_price - fill_close
                    log_trade(current_timestamp_str, position_type, entry_price, entry_rsi,
entry_atr, entry_sl_level, fill_close, "RSI 50 Target", pnl)
                    send_whatsapp_message(f"🎯 SALIDA RSI 50. PNL: {pnl:,.2f}")
                    position_type, position_size_contracts, entry_sl_level = None, 0, 0.0
                    entry_price = 0.0
                    trailing_activated = False
                    last_known_rsi = rsi_signal
                    continue
```

```
        # 5. Predicción LSTM
        prediccion_t_mas_1 = get_lstm_prediction(df_clean, modelo_lstm, scaler_lstm,
N_LAGS)
        if prediccion_t_mas_1 is None:
            logging.warning("Fallo predicción LSTM.")
            last_known_rsi = rsi_signal
            time.sleep(10)
            continue

        log_prediction(current_timestamp_str, current_price, prediccion_t_mas_1, rsi_signal,
atr_signal)

        # 6. Mostrar Estado
        print("\n" + "="*60)
        print(f"[{current_timestamp_str}] BTC 1m | Estado: ACTIVO")
        print(f"Pos: {position_type or 'NINGUNA'} | $ {current_price:,.2f} | RSI(Live):
{rsi_live:.2f} | RSI(Close): {rsi_signal:.2f}")
        print(f"Volatilidad (Lows): {valor_volatilidad_baja:.4f}% | (Highs):
{valor_volatilidad_alta:.4f}%")
        if position_type:
            print(f"Entrada: ${entry_price:,.2f} | SL: ${entry_sl_level:,.2f} (Trailing:
{trailing_activated}) | Meta: RSI 50")

        # 7. ENTRADA (Usamos signal_row)
        final_qty = 0.0
        trigger = False
        new_pos = None

        # Cruce Alcista (Usando RSI Cerrado)
        if (previous_rsi_signal < RSI_BUY_LEVEL and rsi_signal > RSI_BUY_LEVEL) and
position_type != 'LONG':
            logging.info("SEÑAL RSI ALCISTA detectada.")
            if not permitir_compra: logging.warning("Bloqueado por Volatilidad Baja.")
            else:
                trigger, new_pos = True, 'LONG'
                if prediccion_t_mas_1 > current_price * (1 +
LSTM_CONFIRMATION_THRESHOLD):
                    final_qty = BASE_QTY * MODIFIER_HIGH_CONFIDENCE
                else:
                    final_qty = BASE_QTY * MODIFIER_LOW_CONFIDENCE

        # Cruce Bajista (Usando RSI Cerrado)
        elif (previous_rsi_signal > RSI_SELL_LEVEL and rsi_signal < RSI_SELL_LEVEL)
and position_type != 'SHORT':
            logging.info("SEÑAL RSI BAJISTA detectada.")
            if not permitir_venta: logging.warning("Bloqueado por Volatilidad Alta.")
            else:
```

```
                trigger, new_pos = True, 'SHORT'
                if prediccion_t_mas_1 < current_price * (1 -
LSTM_CONFIRMATION_THRESHOLD):
                    final_qty = BASE_QTY * MODIFIER_HIGH_CONFIDENCE
                else:
                    final_qty = BASE_QTY * MODIFIER_LOW_CONFIDENCE

            if trigger:
                qty_contracts = round(final_qty / contract_multiplier)

                if qty_contracts > 0:
                    # Reversión (Si el precio cruza extremo a extremo sin tocar 50, lo cual es raro
pero posible)
                    if position_type:
                        logging.info(f"Reversión: Cerrando {position_type}...")
                        fill_close = place_market_order(client, -position_size_contracts)
                        if fill_close:
                            pnl = (fill_close - entry_price) if position_type == 'LONG' else (entry_price -
fill_close)
                            # Usamos el último entry_sl_level conocido para el log del cierre
                            log_trade(current_timestamp_str, position_type, entry_price, entry_rsi,
entry_atr, entry_sl_level, fill_close, "Reversión", pnl)
                            send_whatsapp_message(f"Reversión ejecutada. PNL: {pnl:,.2f}")
                        else:
                            logging.error("Fallo al cerrar reversión.")
                            last_known_rsi = rsi_signal
                            continue

                    # Apertura (ACTUALIZADO)
                    size = qty_contracts if new_pos == 'LONG' else -qty_contracts
                    fill_open = place_market_order(client, size)
                    if fill_open:
                        # Calcular el Stop Loss basado en ATR
                        sl_diff = atr_signal * ATR_SL_MULTIPLIER

                        if new_pos == 'LONG':
                            new_sl = fill_open - sl_diff
                        else: # SHORT
                            new_sl = fill_open + sl_diff

                        position_type = new_pos
                        position_size_contracts = size
                        entry_price = fill_open
                        entry_sl_level = new_sl # ALMACENAR NIVEL SL
                        entry_rsi, entry_atr = rsi_signal, atr_signal
                        trailing_activated = False # REINICIAR AL ENTRAR
```

```
                logging.info(f"SL Inicial establecido: {entry_sl_level:,.4f} (${sl_diff:,.4f} de búfer)")
                send_whatsapp_message(f"{'🟢' if position_type=='LONG' else '🟣'} {SYMBOL} {position_type} @ ${entry_price:,.2f} | SL: ${entry_sl_level:,.2f}")

            # Actualizar memoria y esperar
            last_known_rsi = rsi_signal
            sleep_time = 60 - (time.time() % 60)
            logging.info(f"Esperando {sleep_time:.1f}s (+2s buffer)...")
            time.sleep(sleep_time + 2)

        except KeyboardInterrupt:
            logging.info("Bot detenido.")
            send_whatsapp_message("🛑 Bot detenido manualmente.")
            break
        except Exception as e:
            logging.error(f"Error en ciclo principal: {e}")
            time.sleep(10)

if __name__ == "__main__":
    main()
```

Informe técnico

Este script es un bot de trading automático para operar en el mercado de futuros de Gate.io operando sobre el BTC/USDT en velas de 1 minuto.

Su lógica de decisión:

1. **Análisis técnico:** Utiliza RSI para entradas en reversión y ATR para gestión de riesgo, agregando Stop loss y volatilidad.
2. **Inteligencia artificial (Predictivo):** Utiliza la red neuronal LSTM para realizar la confirmación de la dirección del precio antes de entrar.

<u>Función de cada línea del script (Líneas 2-14)</u>

gate_api
Esta parte busca tener comunicación con el Exchange.

pandas y numpy
Manipulan los datos financieros.

tensorflow, sklearn
Construcción y procesamiento para el LSTM

pandas_ta

Cálculo eficiente de indicadores técnicos como RSI y ATR.

twilio
Realiza el envió de alertas a Whatsapp

Líneas 17-20
Esta parte del código se encarga de realizar la configuración del sistema de logging para realizar el registro de eventos en la consola y suprimir las advertencias innecesarias de TensorFlow.

Parámetros de Usuario (Líneas 25-83)

Credenciales (Líneas 28-29)
Se utiliza el API_KEY y API_SECRET para realizar el acceso a la cuenta de Gate.

Contrato (Líneas 32-35)

SYMBOL = "BTC_USDT"
Aquí se selecciona el activo con el que se operará

TIMEFRAME = "1m"
Aquí se selecciona la temporalidad de las velas.

LEVERAGE = "10"
Muestra el apalancamiento, en este caso es de 10x

Estrategia (Líneas 38-46)

RSI_PERIOD = 14
Este muestra el periodo estándar del RSI.

RSI_BUY_LEVEL = 30/SELL_LEVEL = 70
Son los umbrales de sobreventa y sobrecompra

RSI_EXIT_TARGET = 50
El bot busca cerrar la operación cuando el RSI regresa al equilibrio, el cual es el RSI de 50.

VOL_LOW_MAX/HIGH_MAX
Son los filtros para evitar operar en el caso de que la volatilidad sea excesiva

ATR_SL_MULTIPLIER = 2.5
Este define el Stop Loss inicial a una distancia de 2.5 veces el ATR.

TRAILING_SL_START_PERCENT = .005
Activa el Stop Loss dinámico (Trailing) cuando la ganancia supera el .5%.

LSTM y entrenamiento (Líneas 49-62)

N_LANGS = 60
Este modelo mira los últimos 60 minutos para predecir el siguiente

CONFIRMATION_THRESHOLD
Este es un filtro mínimo de desviación de precio predicho para aceptar la señal.

Notificaciones (70-75)
Son las credenciales de Twilio para WhatsApp.

Funciones Auxiliares (Líneas 87-230)

Estas funciones son las que actúan como herramientas modulares para el bot.

Setup_log_files (Líneas 87-95)
Crea archivos CSV.

log_prediction y log_trade (Líneas 97-113)
Son las funciones para escribir en los CSV cada vez que el modelo predice o el bot ejecuta una operación.

send_whatsapp_message (Líneas 115-120)
Envía mensajes de WhatsApp.

init_exchange (Líneas 122-137)
Esta se conecta a Gate.io Testnet y ajusta el apalancamiento a 10x automáticamente.

cancel_all_open_orders (Líneas 139-145)
Realiza la limpieza de seguridad y borra órdenes pendientes antes de empezar a operar.

place_market_order (Líneas 154-166)
Realiza envío de órdenes tipo IOC a precio de mercado.

fetch_data_and:indicators (Líneas 175-185)
Descarga las últimas 200 velas y calcula el ATR y RSI usando pandas_ta.

get_lstm_prediction (Líneas 187-196)
Toma los últimos 60 precios de cierre con N_LAGS, los normaliza de escala de 0 a 1 y ejecuta el modelo de LSTM para predecir el precio de cierre de la siguiente vela (t + 1)

Entrenamiento del modelo IA (Líneas 233-269)

Esta sección realiza la ejecución en el caso de que no existan los archivos del modelo.

fetch_training_data
Descarga las 2,000 velas históricas.

train_and_save_model

Normaliza los datos con MinMaxScaler, crea estructuras de datos X (pasado) e y (futuro)

Busca definir la arquitectura de la Red Neuronal con LSTM (50 neuronas) la cual es la capa recurrente para detectar patrones temporales, el dropout (0.2) que apaga el 20% de neuronas al azar para evitar sobreajuste y Dense que son capas finales para condensar la predicción a un solo valor. Por último entrena el modelo por 20 épocas y lo guarda en .h5

Bucle principal (Líneas 274-464)

Es la parte que realiza el cerebro del bot en bucle infinito

Inicialización (Líneas 275-302)
Busca cargar el API, Logs, Modelo de la IA y obtener detalles del contrato.

Variables del Estado (Líneas 290-296)

entry_sl_level
Guarda el precio exacto del Stop Loss.

trailing_activated
Busca (True/False) para saber si el SL ya se está moviendo.

Bucle (Línea 304 en adelante)

Sincronización (Líneas 308 a la 318)
Revisa si la posición en el exchange coincide con la memoria del bot. si se cierra manualmente resetea las variables.

Cálculo de volatilidad (Línea 324 a la 329)
Realiza la medición estándar de los cambios porcentuales de Low y High para filtrar mercados ruidosos.

Lógica de Trailing Stop (354-387)

Realiza el cálculo del PNL actual, en el caso de que el PNL>.5% (TRAILING_SL_START_PERCENT) activa el flag trailing_activated. En el caso de que esté activo, mueve el SL a Precio Actual - (2.5 * ATR). Solo lo mueve a favor, nunca retrocede.

Verificación de salidas (Líneas 391-434)

Stop Loss
SI el precio toca entry_sl_level cierra la posición y envía la alerta

Take profit (RSI 50)
Si el RSI toca 50 (retorno a la media), cierra la posición en ganancia.

Predicción (Líneas 437-442)
Consulta el LSTM para saber la proyección del precio.

Lógica de entrada (Líneas 451-476)

Detecta cruces de RSI (de abajo hacia arriba en 30 para LONG, de arriba hacia abajo en 70 para SHORT), verifica los filtros de volatilidad, realiza la confirmación de LSTM. En el caso de que la señal dea LONG, la IA debe predecir que el precio subirá. Si es SHORT, debe predecir que bajará.

Por otro lado, ajusta el tamaño de la posición (BASE_QTY) según la confianza de la predicción.

Ejecución de entrada (Líneas 478-514)

Envía la orden al mercado, calcula el Stop Loss inicial con Precio Entrada +- (ATR*2.5). Reinicia el trailing_activated a False y envía una notificación de WhatsApp con el precio y el SL definido.

Espera (Líneas 517-519)
Duerme el script hasta el inicio del siguiente minuto para evitar el spam a la API.

Código de honor: Damos nuestra palabra de que hemos realizado esta actividad con integridad académica.