ChatGPT

```
using System;
using System.IO;
using System.Linq;
using cAlgo.API;
using cAlgo.API.Internals;
using cAlgo.API.Indicators;

namespace cAlgo.Robots
{
    // Main trading robot class combining multiple strategies and controls
    [Robot(TimeZone = TimeZones.UTC, AccessRights = AccessRights.FullAccess)]
    public class MetaVvAIBot : Robot
    {
        // ====================== GENERAL CONFIGURATION PARAMETERS
======================
        [Parameter("Enable Ultra Mode (fast scalping)", Group = "Modes",
DefaultValue = true)]
        public bool UltraModeEnabled { get; set; }

        [Parameter("Enable Precision Mode (conservative)", Group = "Modes",
DefaultValue = true)]
        public bool PrecisionModeEnabled { get; set; }

        [Parameter("Enable MetaVvAI Controller (consensus)", Group = "Modes",
DefaultValue = false)]
        public bool MetaControllerEnabled { get; set; }

        [Parameter("Volume in Lots", Group = "Trade", DefaultValue = 1.0)]
        public double VolumeInLots { get; set; }

        [Parameter("Tick Volume (Lots)", Group = "Trade", DefaultValue = 1.0)]
        public double TickVolumeInLots { get; set; }

        [Parameter("Ultra Bot Label", Group = "Trade", DefaultValue =
"UltraModeBot")]
        public string UltraLabel { get; set; }

        [Parameter("Precision Bot Label", Group = "Trade", DefaultValue =
"PrecisionModeBot")]
        public string PrecisionLabel { get; set; }

        // ====================== RSI FILTER PARAMETERS ======================
        [Parameter("Enable RSI Filter", Group = "Filters", DefaultValue = true)]
        public bool RSIFilterEnabled { get; set; }
```

```
        [Parameter("RSI Period", Group = "Filters", DefaultValue = 14)]
        public int RSIPeriod { get; set; }

        [Parameter("RSI Overbought Level", Group = "Filters", DefaultValue =
70)]
        public int RSIOverbought { get; set; }

        [Parameter("RSI Oversold Level", Group = "Filters", DefaultValue = 30)]
        public int RSIoversold { get; set; }

        // ======================== BOLLINGER BANDS PARAMETERS
========================
        [Parameter("Enable Bollinger Filter", Group = "Filters", DefaultValue =
true)]
        public bool BollingerFilterEnabled { get; set; }

        [Parameter("Bollinger Band Period", Group = "Filters", DefaultValue =
20)]
        public int BollingerPeriod { get; set; }

        [Parameter("Bollinger Std Dev", Group = "Filters", DefaultValue = 2.0)]
        public double BollingerStdDev { get; set; }

        // ======================== ATR STOP/TP PARAMETERS
========================
        [Parameter("Enable ATR SL/TP", Group = "Stops", DefaultValue = true)]
        public bool ATRStopTPEnabled { get; set; }

        [Parameter("ATR Period", Group = "Stops", DefaultValue = 14)]
        public int ATRPeriod { get; set; }

        [Parameter("ATR SL/TP Multiplier", Group = "Stops", DefaultValue = 2.0)]
        public double ATRMultiplier { get; set; }

        // ======================== VOLUME FILTER PARAMETERS
========================
        [Parameter("Enable Volume Filter", Group = "Filters", DefaultValue =
true)]
        public bool VolumeFilterEnabled { get; set; }

        [Parameter("Minimum Tick Volume", Group = "Filters", DefaultValue =
100)]
        public int MinTickVolume { get; set; }

        // ======================== SCHEDULE FILTER PARAMETERS
========================
        [Parameter("Enable Schedule Filter", Group = "Schedule", DefaultValue =
```

```
false)]
        public bool ScheduleFilterEnabled { get; set; }

        [Parameter("Trade Monday", Group = "Schedule", DefaultValue = true)]
        public bool TradeMonday { get; set; }

        [Parameter("Trade Tuesday", Group = "Schedule", DefaultValue = true)]
        public bool TradeTuesday { get; set; }

        [Parameter("Trade Wednesday", Group = "Schedule", DefaultValue = true)]
        public bool TradeWednesday { get; set; }

        [Parameter("Trade Thursday", Group = "Schedule", DefaultValue = true)]
        public bool TradeThursday { get; set; }

        [Parameter("Trade Friday", Group = "Schedule", DefaultValue = true)]
        public bool TradeFriday { get; set; }

        [Parameter("Start Hour (UTC)", Group = "Schedule", DefaultValue = 0)]
        public int TradeStartHour { get; set; }

        [Parameter("End Hour (UTC)", Group = "Schedule", DefaultValue = 23)]
        public int TradeEndHour { get; set; }

        // ======================== ADAPTIVE LEARNER PARAMETERS
========================
        [Parameter("Enable Adaptive Learning", Group = "Adaptive", DefaultValue
= false)]
        public bool AdaptiveLearningEnabled { get; set; }

        [Parameter("Adaptive: Learning Window (trades)", Group = "Adaptive",
DefaultValue = 50)]
        public int AdaptiveWindow { get; set; }

        // ======================== LOGGING PARAMETERS ========================
        [Parameter("Enable CAB Logs", Group = "Logging", DefaultValue = false)]
        public bool UseCABLogs { get; set; }

        [Parameter("Log File Name", Group = "Logging", DefaultValue =
"CABLog.txt")]
        public string LogFileName { get; set; }

        // ======================== INTERNAL INDICATORS ========================
        private RelativeStrengthIndex _rsi;
        private BollingerBands _bollinger;
        private AverageTrueRange _atr;
        private TickVolume _tickVolume;
```

```
        // ======================= STATS FOR ADAPTIVE LEARNING
=======================
        private double _ultraProfitTotal = 0;
        private int _ultraTradesCount = 0;
        private double _precisionProfitTotal = 0;
        private int _precisionTradesCount = 0;

        // ======================= MISCELLANEOUS ======================
        private double _volumeInUnits;
        private double _tickVolumeInUnits;

        /// <summary>
        /// Called when the cBot is started. Initialize indicators, logging,
etc.
        /// </summary>
        protected override void OnStart()
        {
            // Convert lot parameters to volume in units
            _volumeInUnits = Symbol.QuantityToVolumeInUnits(VolumeInLots);
            _tickVolumeInUnits =
Symbol.QuantityToVolumeInUnits(TickVolumeInLots);

            // Initialize indicators based on parameters
            if (RSIFilterEnabled)
                _rsi = Indicators.RelativeStrengthIndex(MarketSeries.Close,
RSIPeriod);

            if (BollingerFilterEnabled)
                _bollinger = Indicators.BollingerBands(MarketSeries.Close,
BollingerPeriod, BollingerStdDev, MovingAverageType.Simple);

            if (ATRStopTPEnabled)
                _atr = Indicators.AverageTrueRange(ATRPeriod,
MovingAverageType.Simple);

            if (VolumeFilterEnabled)
                _tickVolume = Indicators.TickVolume();

            // Initialize CAB logging if enabled
            if (UseCABLogs)
                Logger.Init(LogFileName);

            // Log start event
            Print("MetaVvAI Bot Started with modes - Ultra: {0}, Precision:
{1}", UltraModeEnabled, PrecisionModeEnabled);
            if (UseCABLogs) Logger.Log("Bot started at " +
Server.Time.ToString("u"));
        }
```

```
        /// <summary>
        /// Called on each new bar (based on the selected timeframe). Primary
trading logic is executed here.
        /// </summary>
        protected override void OnBar()
        {
            DateTime now = MarketSeries.OpenTime.LastValue;  // Current bar's
open time

            // ======================== SCHEDULE CHECK ========================
            if (ScheduleFilterEnabled)
            {
                // Day of week filter
                bool dayAllowed = (now.DayOfWeek == DayOfWeek.Monday &&
TradeMonday) ||
                                  (now.DayOfWeek == DayOfWeek.Tuesday &&
TradeTuesday) ||
                                  (now.DayOfWeek == DayOfWeek.Wednesday &&
TradeWednesday) ||
                                  (now.DayOfWeek == DayOfWeek.Thursday &&
TradeThursday) ||
                                  (now.DayOfWeek == DayOfWeek.Friday &&
TradeFriday);
                // Time of day filter
                bool timeAllowed = now.TimeOfDay >=
TimeSpan.FromHours(TradeStartHour) &&
                                   now.TimeOfDay <
TimeSpan.FromHours(TradeEndHour);
                if (!dayAllowed || !timeAllowed)
                {
                    // Skip trading outside allowed schedule
                    return;
                }
            }

            // ======================== DETERMINE SIGNALS ========================
            bool ultraBuySignal = false;
            bool ultraSellSignal = false;
            bool precisionBuySignal = false;
            bool precisionSellSignal = false;

            // Example Ultra Mode condition: momentum-based (simple price rise/
fall)
            if (Bars.ClosePrices.Count > 2)  // Ensure sufficient history
            {
                // Check simple price direction
                double lastClose = Bars.ClosePrices.Last(1);
```

```
                double prevClose = Bars.ClosePrices.Last(2);

                if (lastClose > prevClose)
                    ultraBuySignal = true;
                else if (lastClose < prevClose)
                    ultraSellSignal = true;
            }

            // Example Precision Mode conditions: multiple indicator
confirmations
            // Only evaluate if relevant filters are enabled
            if (RSIFilterEnabled && BollingerFilterEnabled &&
Bars.ClosePrices.Count > 0)
            {
                double rsiValue = _rsi.Result.LastValue;
                double currentPrice = Bars.ClosePrices.Last(1);
                double upperBand = _bollinger.Top.Last(1);
                double lowerBand = _bollinger.Bottom.Last(1);

                // Precision Buy: price near lower Bollinger band and RSI
oversold
                if (currentPrice <= lowerBand && rsiValue < RSIoversold)
                    precisionBuySignal = true;

                // Precision Sell: price near upper Bollinger band and RSI
overbought
                if (currentPrice >= upperBand && rsiValue > RSIOverbought)
                    precisionSellSignal = true;
            }

            // Volume filter check: (applies to both modes if enabled)
            bool volumeOk = true;
            if (VolumeFilterEnabled && _tickVolume != null)
            {
                // Use last closed bar tick volume
                double lastTickVol = _tickVolume.Result.Last(1);
                volumeOk = lastTickVol >= MinTickVolume;
            }

            // ======================== TRADING LOGIC ========================
            // ULTRA MODE TRADING
            if (UltraModeEnabled)
            {
                // Evaluate buy
                if (ultraBuySignal && volumeOk)
                {
                    // If Meta controller is enabled with Precision, require
Precision also signals buy
```

```
                    if (!MetaControllerEnabled || !PrecisionModeEnabled ||
precisionBuySignal)
                    {
                        // Place buy order with ATR-based stop and profit
targets
                        double sl = 0, tp = 0;
                        if (ATRStopTPEnabled)
                        {
                            double atrValue = _atr.Result.LastValue;
                            double atrPips = atrValue / Symbol.PipSize;
                            sl = atrPips * ATRMultiplier;
                            tp = atrPips * ATRMultiplier;
                        }
                        var result = ExecuteMarketOrder(TradeType.Buy,
SymbolName, _volumeInUnits, UltraLabel, sl, tp);
                        if (result.IsSuccessful)
                        {
                            if (UseCABLogs) Logger.Log($"Ultra BUY executed at
{result.Position.EntryPrice:F5}, SL={sl:F1}, TP={tp:F1}");
                        }
                    }
                }
                // Evaluate sell
                if (ultraSellSignal && volumeOk)
                {
                    if (!MetaControllerEnabled || !PrecisionModeEnabled ||
precisionSellSignal)
                    {
                        double sl = 0, tp = 0;
                        if (ATRStopTPEnabled)
                        {
                            double atrValue = _atr.Result.LastValue;
                            double atrPips = atrValue / Symbol.PipSize;
                            sl = atrPips * ATRMultiplier;
                            tp = atrPips * ATRMultiplier;
                        }
                        var result = ExecuteMarketOrder(TradeType.Sell,
SymbolName, _volumeInUnits, UltraLabel, sl, tp);
                        if (result.IsSuccessful)
                        {
                            if (UseCABLogs) Logger.Log($"Ultra SELL executed at
{result.Position.EntryPrice:F5}, SL={sl:F1}, TP={tp:F1}");
                        }
                    }
                }
            }

            // PRECISION MODE TRADING
```

```
            if (PrecisionModeEnabled)
            {
                // Evaluate buy
                if (precisionBuySignal && volumeOk)
                {
                    if (!MetaControllerEnabled || !UltraModeEnabled ||
ultraBuySignal)
                    {
                        double sl = 0, tp = 0;
                        if (ATRStopTPEnabled)
                        {
                            double atrValue = _atr.Result.LastValue;
                            double atrPips = atrValue / Symbol.PipSize;
                            sl = atrPips * ATRMultiplier;
                            tp = atrPips * ATRMultiplier;
                        }
                        var result = ExecuteMarketOrder(TradeType.Buy,
SymbolName, _volumeInUnits, PrecisionLabel, sl, tp);
                        if (result.IsSuccessful)
                        {
                            if (UseCABLogs) Logger.Log($"Precision BUY executed
at {result.Position.EntryPrice:F5}, SL={sl:F1}, TP={tp:F1}");
                        }
                    }
                }
                // Evaluate sell
                if (precisionSellSignal && volumeOk)
                {
                    if (!MetaControllerEnabled || !UltraModeEnabled ||
ultraSellSignal)
                    {
                        double sl = 0, tp = 0;
                        if (ATRStopTPEnabled)
                        {
                            double atrValue = _atr.Result.LastValue;
                            double atrPips = atrValue / Symbol.PipSize;
                            sl = atrPips * ATRMultiplier;
                            tp = atrPips * ATRMultiplier;
                        }
                        var result = ExecuteMarketOrder(TradeType.Sell,
SymbolName, _volumeInUnits, PrecisionLabel, sl, tp);
                        if (result.IsSuccessful)
                        {
                            if (UseCABLogs)
Logger.Log($"Precision SELL executed at {result.Position.EntryPrice:F5},
SL={sl:F1}, TP={tp:F1}");
                        }
                    }
```

```
                }
            }

            // End of OnBar logic
        }

        /// <summary>
        /// Called whenever a position is closed. Used for logging and adaptive
learning updates.
        /// </summary>
        /// <param name="args">Position closed event arguments containing
details.</param>
        protected override void OnPositionClosed(PositionClosedEventArgs args)
        {
            // Calculate profit of the trade (in quote currency)
            var profit = args.Position.GrossProfit;
            bool isUltra = args.Position.Label == UltraLabel;
            bool isPrecision = args.Position.Label == PrecisionLabel;

            // Log trade result
            string mode = isUltra ? "Ultra" : (isPrecision ? "Precision" :
"Unknown");
            string message =
$"{mode} Trade closed. P/L: {profit:F2}, Direction: {args.Position.TradeType},
Entry: {args.Position.EntryPrice:F5}, Exit: {args.Position.ClosePrice:F5}";
            Print(message);
            if (UseCABLogs) Logger.Log(message);

            // Update stats for adaptive learning
            if (AdaptiveLearningEnabled)
            {
                if (isUltra)
                {
                    _ultraTradesCount++;
                    _ultraProfitTotal += profit;
                    // Check if learning window reached
                    if (_ultraTradesCount >= AdaptiveWindow)
                    {
                        // Evaluate performance and adjust RSI thresholds if
needed
                        if (_ultraProfitTotal < 0)
                        {
                            // If losing, make RSI oversold smaller (more
conservative)
                            RSIoversold = Math.Max(RSIoversold - 5, 5);
                            RSIOverbought = Math.Min(RSIOverbought + 5, 95);
                            if (UseCABLogs) Logger.Log($"Adaptive Ultra:
Adjusted RSI oversold to {RSIoversold}, overbought to {RSIOverbought}");
```

```
                        }
                        else
                        {
                            // If profitable, fine-tune to try for more trades
                            RSIoversold = Math.Min(RSIoversold + 5, 50);
                            RSIOverbought = Math.Max(RSIOverbought - 5, 50);
                            if (UseCABLogs) Logger.Log($"Adaptive Ultra:
Adjusted RSI oversold to {RSIoversold}, overbought to {RSIOverbought}");
                        }
                        // Reset counters
                        _ultraTradesCount = 0;
                        _ultraProfitTotal = 0;
                    }
                }
                else if (isPrecision)
                {
                    _precisionTradesCount++;
                    _precisionProfitTotal += profit;
                    if (_precisionTradesCount >= AdaptiveWindow)
                    {
                        if (_precisionProfitTotal < 0)
                        {
                            RSIoversold = Math.Max(RSIoversold - 3, 5);
                            RSIOverbought = Math.Min(RSIOverbought + 3, 95);
                            if (UseCABLogs) Logger.Log($"Adaptive Precision:
Adjusted RSI oversold to {RSIoversold}, overbought to {RSIOverbought}");
                        }
                        else
                        {
                            RSIoversold = Math.Min(RSIoversold + 3, 50);
                            RSIOverbought = Math.Max(RSIOverbought - 3, 50);
                            if (UseCABLogs) Logger.Log($"Adaptive Precision:
Adjusted RSI oversold to {RSIoversold}, overbought to {RSIOverbought}");
                        }
                        _precisionTradesCount = 0;
                        _precisionProfitTotal = 0;
                    }
                }
            }
        }
    }

    /// <summary>
    /// CAB Logging utility class to log events and diagnostics to a file.
    /// </summary>
    public static class Logger
    {
        private static string _filePath;
```

```
        /// <summary>
        /// Initializes the logger by setting the log file path.
        /// </summary>
        public static void Init(string fileName)
        {
            // Use the default cAlgo data directory to store the log file
            _filePath = fileName;
            // Create or clear existing file at start
            try
            {
                File.WriteAllText(_filePath, "CAB Log Initialized at " +
DateTime.UtcNow.ToString("u") + "\n");
            }
            catch (Exception ex)
            {
                // Unable to write log file
            }
        }

        /// <summary>
        /// Logs a message with timestamp to the log file.
        /// </summary>
        public static void Log(string message)
        {
            if (string.IsNullOrEmpty(_filePath))
                return;

            string line = $"{DateTime.UtcNow.ToString(\"u\")} - {message}\n";
            try
            {
                File.AppendAllText(_filePath, line);
            }
            catch (Exception ex)
            {
                // Ignore logging errors
            }
        }
    }
}
```