Skill ML.NET Anomaly Detection

Detect anomalies in a single sensor tag using ML.NET SSA (Singular Spectrum Analysis). Supports: **Spike Detection** for sudden outliers. The AI generates the full C# Script Class, connects it to a live tag, creates output tags, and configures model persistence.

AI Integration Platform Skills Library Skill ML.NET Skill ML.NET Anomaly Detection

- When to Use This Skill
- Prerequisites
- MCP Tools and Tables
- Step 1: Create Output Tags
- Step 2: Create the Script Class
  - ML.NET Namespace Declaration
  - Tag References Inside Script Classes
  - SSA Spike Detection Pipeline
  - Full Class Example — Anomaly Detection (Spike)
- Step 3: Write the Class via MCP
- Step 4: Create the Trigger
  - Expression (OnChange) — triggers on each tag value change
  - ServerStartup — always wire LoadModel
- Step 5: Verify
- Common Pitfalls
- See also

When to Use This Skill

- User chose **Anomaly Detection — SSA Spike** in the ML.NET router skill (Step 0)
- Monitoring a **single sensor** for outliers, spikes, drift, or regime changes
- User goals: predictive maintenance (single sensor), detect sensor failures, detect gradual process shift

Prerequisites

- Solution open with the input tag already created and receiving data (live or simulated)
- Solution target platform set to **Multiplatform** (ML.NET requires .NET 8+)
- The user has confirmed: (1) which algorithm variant (Spike), (2) which tag to monitor, (3) output folder path

MCP Tools and Tables

Category	Items
Tools	`get_table_schema`, `write_objects`, `get_objects`
Tables	`UnsTags`, `ScriptsClasses`, `ScriptsExpressions`, `ScriptsTasks`

Step 1: Create Output Tags

Create tags to receive the model's predictions. Place them under a `/ML/` subfolder relative to the input tag's asset path.

```
get_table_schema('UnsTags')
```

```
{
  "table_type": "UnsTags",
  "data": [
    { "Name": "<AssetPath>/ML/AnomalyScore", "DataType": "Double", "Description": "Anomaly score (0=normal,
higher=anomalous)" },
    { "Name": "<AssetPath>/ML/IsAnomaly", "DataType": "Boolean", "Description": "True when anomaly detected"
},
    { "Name": "<AssetPath>/ML/LastPrediction", "DataType": "DateTime", "Description": "Timestamp of last
prediction" }
  ]
}
```

Replace `<AssetPath>` with the actual asset folder path (e.g., `Plant/Reactor1`).

Step 2: Create the Script Class

ML.NET Namespace Declaration

Script Classes using ML.NET require the `NamespaceDeclarations` field. ML.NET is pre-installed with FrameworX — no NuGet packages needed.

> **Critical:** The field `AddMLNetNamespaces` does **not exist** and is silently ignored. Always use `NamespaceDeclarations` with a semicolon-separated string. Omitting it causes `CS0246` / `CS0234` compilation errors.

```
"NamespaceDeclarations": "Microsoft.ML;Microsoft.ML.Data;Microsoft.ML.Transforms;Microsoft.ML.Transforms.
TimeSeries;Microsoft.ML.Transforms.Text;Microsoft.ML.Trainers;Microsoft.ML.TimeSeries"
```

Tag References Inside Script Classes

Always use the `@Tag.` prefix to read or write tag values:

```
// Read from a tag
double temp = @Tag.Plant/Reactor1/Temperature.Value;

// Write to a tag
@Tag.Plant/Reactor1/ML/AnomalyScore.Value = score;
@Tag.Plant/Reactor1/ML/IsAnomaly.Value = true;
@Tag.Plant/Reactor1/ML/LastPrediction.Value = DateTime.Now;
```

Important: ML.NET expects `float` but FrameworX tags use `double`. Always cast with `(float)` when feeding ML.NET and cast back to `double` when writing to tags.

SSA Spike Detection Pipeline

Use for sudden outliers, spikes, abnormal readings (e.g., pressure spikes, vibration bursts, temperature jumps).

```
var pipeline = mlContext.Transforms.DetectSpikeBySsa(
    outputColumnName: "Prediction",
    inputColumnName: nameof(SensorData.Value),
    confidence: 95.0,
    pvalueHistoryLength: 10,
    trainingWindowSize: 100,
    seasonalityWindowSize: 10);
```

Output: `double[] Prediction` with `[0]=isAnomaly, [1]=score, [2]=pValue`

Full Class Example — Anomaly Detection (Spike)

```
public class SensorData
{
    public float Value { get; set; }
}

public class SpikePrediction
{
    [VectorType(3)]
    public double[] Prediction { get; set; }
}

private static MLContext mlContext = new MLContext(seed: 0);
private static ITransformer model;
private static IDataView lastTrainingDataView;
private static bool modelTrained = false;
private static List<SensorData> trainingBuffer = new List<SensorData>();
private const int MinTrainingSize = 100;
private static readonly string ModelPath = Path.Combine(@Info.GetExecutionPath(), "<ClassName>.mlnet");

public int Predict(double inputValue)
{
    trainingBuffer.Add(new SensorData { Value = (float)inputValue });

    if (!modelTrained && trainingBuffer.Count >= MinTrainingSize)
        TrainModel();
```

```
    if (modelTrained)
        RunPrediction();

        return 1;
}

public void LoadModel()
{
    if (File.Exists(ModelPath))
    {
        model = mlContext.Model.Load(ModelPath, out _);
        modelTrained = true;
    }
}

private void TrainModel()
{
    lastTrainingDataView = mlContext.Data.LoadFromEnumerable(trainingBuffer);
    var pipeline = mlContext.Transforms.DetectSpikeBySsa(
        outputColumnName: "Prediction",
        inputColumnName: nameof(SensorData.Value),
        confidence: 95.0,
        pvalueHistoryLength: 10,
        trainingWindowSize: 100,
        seasonalityWindowSize: 10);

    model = pipeline.Fit(lastTrainingDataView);
    modelTrained = true;
    SaveModel();
}

private void SaveModel()
{
    mlContext.Model.Save(model, lastTrainingDataView.Schema, ModelPath);
}

private void RunPrediction()
{
    var dataView = mlContext.Data.LoadFromEnumerable(trainingBuffer);
    var transformed = model.Transform(dataView);
    var predictions = mlContext.Data.CreateEnumerable<SpikePrediction>(transformed, reuseRowObject: false).
ToList();
    var latest = predictions.Last();

    @Tag.<AssetPath>/ML/IsAnomaly.Value = (int)latest.Prediction[0];
    @Tag.<AssetPath>/ML/AnomalyScore.Value = latest.Prediction[1];
    @Tag.<AssetPath>/ML/LastPrediction.Value = DateTime.Now;
}
```

Step 3: Write the Class via MCP

```
get_table_schema('ScriptsClasses')
```

```
{
  "table_type": "ScriptsClasses",
  "data": [
    {
      "Name": "<ClassName>",
      "Code": "CSharp",
      "Domain": "Server",
      "ClassContent": "Methods",
      "NamespaceDeclarations": "Microsoft.ML;Microsoft.ML.Data;Microsoft.ML.Transforms;Microsoft.ML.
Transforms.TimeSeries;Microsoft.ML.Transforms.Text;Microsoft.ML.Trainers;Microsoft.ML.TimeSeries",
      "Contents": "<AI-generated C# code from full class example above>"
    }
  ]
}
```

> Field names matter: the language field is `Code` (not `Language`), and the code body field is `Contents` (not `Code`). Using wrong field names results in silent data loss.

Step 4: Create the Trigger

Expression (OnChange) — triggers on each tag value change

```
get_table_schema('ScriptsExpressions')
```

```
{
  "table_type": "ScriptsExpressions",
  "data": [
    {
      "Name": "ML_Predict_<SensorName>",
      "ObjectName": "",
      "Expression": "@Script.Class.<ClassName>.Predict(@Tag.<AssetPath>.<Member>)",
      "Execution": "OnChange",
      "Trigger": ""
    }
  ]
}
```

> **`ObjectName` must be empty** when the ML class writes prediction results to output tags internally (inside `RunPrediction()`). Setting a non-empty `ObjectName` on a void method call causes a type assignment error.
>
> **Use `Trigger` (not `TriggerTag`)** — `TriggerTag` is not a valid field and is silently ignored, causing the expression to never fire. `Trigger` accepts the tag path without the `Tag.` prefix.

ServerStartup — always wire LoadModel

Read the existing `ServerStartup` task first (document object — read-modify-write), then append:

```
Script.Class.<ClassName>.LoadModel();
```

Step 5: Verify

1. **Confirm Multiplatform** — ML.NET requires .NET 8+. If the solution targets Windows (.NET 4.8), training will fail with a `System.Math` / `CpuMath` error. Instruct the user: *"Before starting the runtime, please confirm your solution is set to Multiplatform: Solution Settings Target Platform = Multiplatform, then Product Modify."*
2. **Do NOT start the runtime automatically.** Inform the user that all scripts are configured and they can start the runtime when ready.
3. **Wait for training** — the model needs `MinTrainingSize` data points (default 100) before predictions begin
4. **Check output tags** — verify `LastPrediction` timestamp is updating

Common Pitfalls

Mistake	Why It Happens	How to Avoid
Missing ML.NET namespaces	Used `AddMLNetNamespaces: true` (field does not exist)	Always set `NamespaceDeclarations` with the full semicolon-separated string
`CS0234` error in ScriptsTasks	Called `Script.Class.Name.Predict()` without `@` prefix	Always use `@Script.Class.Name.Predict()`
Non-empty `ObjectName` on void Predict()	Expression tries to assign void return to a tag	Leave `ObjectName` empty
Used `TriggerTag` field	Field does not exist — silently ignored	Use `Trigger` field
`CS0029` bool-to-int on Digital tag	Digital tags are `int`, not `bool`	Use ternary: `@Tag.Path.Value = (condition) ? 1 : 0;`
`CS0246` Script Class not found	Class compiles after Task that references it	Set `BuildOrder: "1"` on the ML Script Class
`System.Math` load error at training	Solution targets .NET 4.8	Switch to Multiplatform (.NET 8+)
Wrong data types	ML.NET expects `float`, tags are `double`	Cast with `(float)` for ML.NET, `(double)` for tags
Model lost on restart	SaveModel or LoadModel not wired	Always include both + wire LoadModel in ServerStartup

See also

- Skill ML.NET. Router skill that selects the appropriate ML.NET sub-skill based on user goals.
- Skill ML.NET Forecasting. Sister sub-skill for SSA time-series forecasting on a single sensor tag.
- Skill ML.NET Regression. Sister sub-skill for FastTree regression from multiple feature tags.
- Skill ML.NET Classification. Sister sub-skill for FastTree binary classification from multiple feature tags.