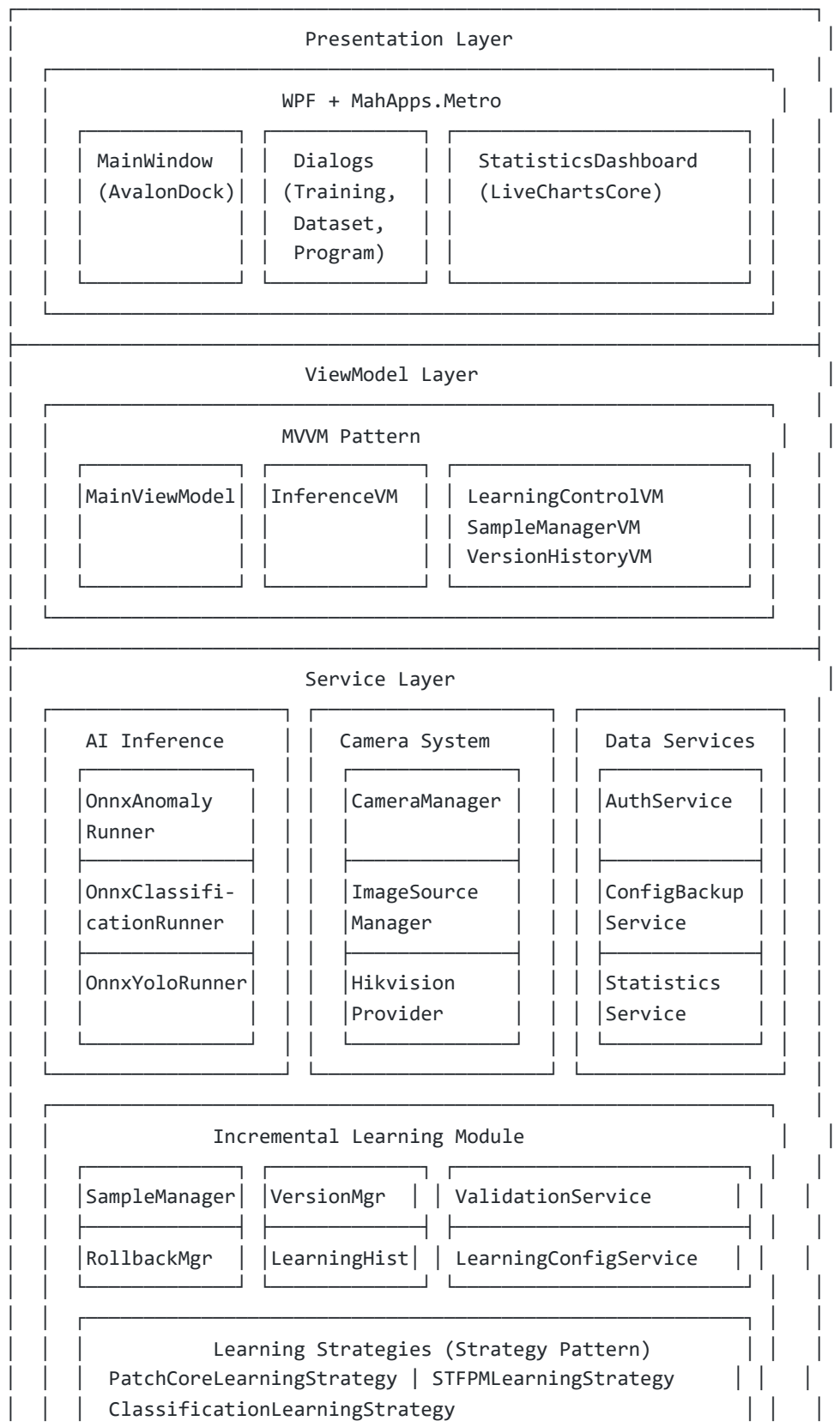
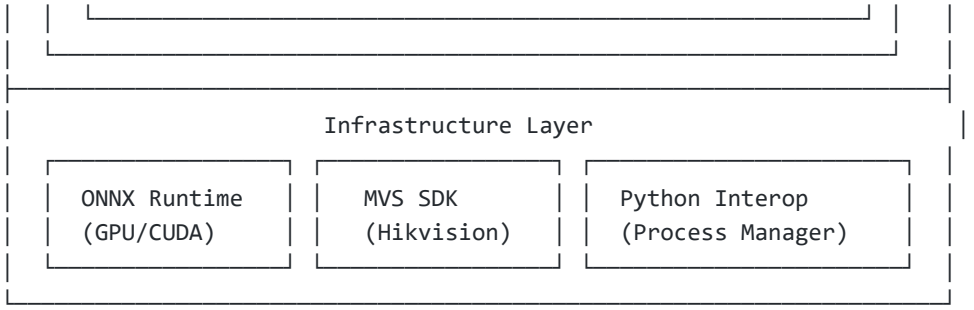
🏗️ 系统架构文档

本文档详细介绍 AI Vision Inspector 的技术架构，适合面试时深入讨论。

整体架构





核心设计模式

1. MVVM (Model-View-ViewModel)

```
// ViewModel 示例
public class MainViewModel : ViewModelBase
{
    private readonly IInferenceService _inferenceService;

    public ICommand RunInferenceCommand { get; }

    public MainViewModel(IInferenceService inferenceService)
    {
        _inferenceService = inferenceService;
        RunInferenceCommand = new RelayCommand(ExecuteInference);
    }
}
```

优势:

- 视图与业务逻辑分离
- 便于单元测试
- 支持数据绑定

2. 策略模式 (Strategy Pattern)

```
// 学习策略接口
public interface ILearningStrategy
{
    Task<LearningResult> ExecuteAsync(
        string modelPath,
        IEnumerable<FeedbackSample> samples,
        CancellationToken ct);
}

// 具体策略实现
public class PatchCoreLearningStrategy : ILearningStrategy { }
public class STFPMLearningStrategy : ILearningStrategy { }
public class ClassificationLearningStrategy : ILearningStrategy { }

// 策略工厂
```

```
public class LearningStrategyFactory
{
    public ILearningStrategy Create(string taskType)
    {
        return taskType switch
        {
            "anomaly_patchcore" => new PatchCoreLearningStrategy(),
            "anomaly_stfpm" => new STFPMLearningStrategy(),
            "classification" => new ClassificationLearningStrategy(),
            _ => throw new NotSupportedException()
        };
    }
}
```

优势：

- 算法可替换
- 符合开闭原则
- 便于扩展新算法

3. 服务定位器 (Service Locator)

```
public static class ServiceLocator
{
    private static readonly Dictionary<Type, object> _services = new();

    public static void Register<T>(T service) where T : class
    {
        _services[typeof(T)] = service;
    }

    public static T Get<T>() where T : class
    {
        return (T)_services[typeof(T)];
    }
}
```

4. 接口抽象

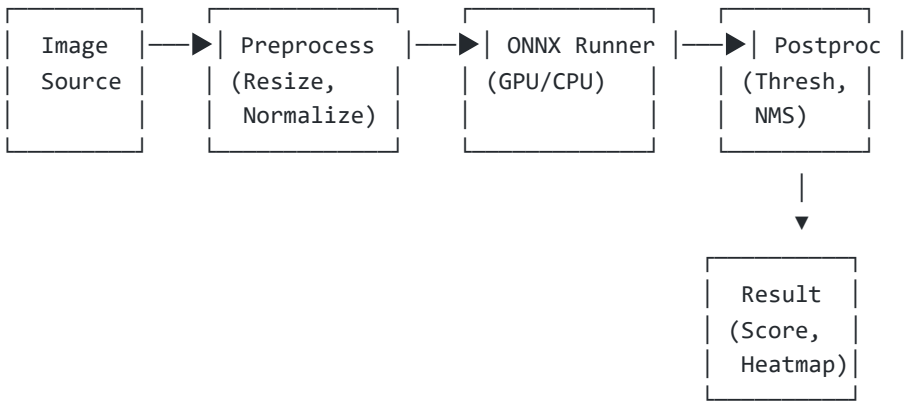
```
// 图像源抽象
public interface IImageSource
{
    Task<BitmapSource> GetImageAsync();
    event EventHandler<ImageAcquiredEventArgs> ImageAcquired;
}

// 相机提供者抽象
public interface ICameraProvider
{
    IEnumerable<CameraInfo> EnumerateDevices();
    ICameraDevice Connect(CameraInfo info);
}
```

优势：

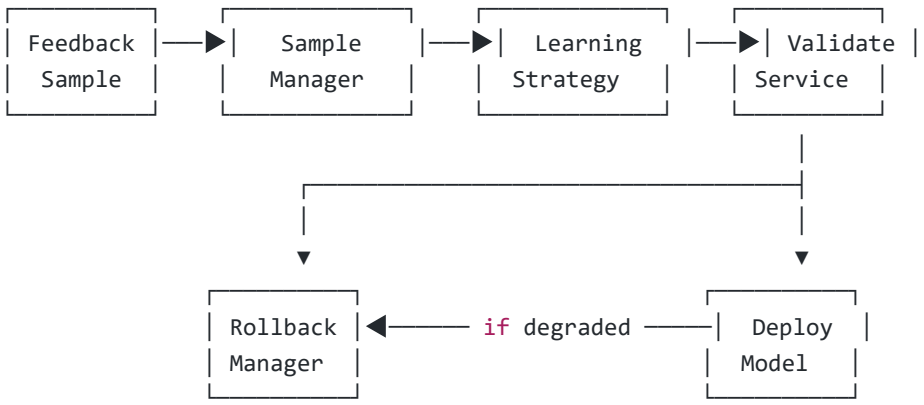
- 解耦具体实现
- 便于 Mock 测试
- 支持多品牌相机扩展

数据流

推理流程



增量学习流程



关键技术决策

1. 为什么选择 ONNX Runtime?

方案	优点	缺点
ONNX Runtime ✅	跨框架、GPU 加速、C# 原生支持	需要模型转换
PyTorch C++	原生支持	部署复杂、包体积大

方案	优点	缺点
TensorRT	极致性能	NVIDIA 专属、模型兼容性

2. 为什么选择 WPF 而非 Web?

方案	优点	缺点
WPF ✅	原生性能、硬件访问、离线运行	仅 Windows
Electron	跨平台、Web 技术栈	性能开销、相机 SDK 集成困难
Web	远程访问、跨平台	实时性差、硬件访问受限

3. 增量学习策略选择

模型类型	更新策略	原因
PatchCore	Memory Bank 更新	无需重训 Backbone
STFPM	Student 网络微调	保持 Teacher 不变
分类模型	分类头微调	冻结 Backbone 防止遗忘

性能优化

1. GPU 加速

```
var sessionOptions = new SessionOptions();
sessionOptions.AppendExecutionProvider_CUDA(0);
_session = new InferenceSession(modelPath, sessionOptions);
```

2. 内存管理

- 使用 `using` 语句及时释放 Tensor
- 图像处理使用 `Span<T>` 避免拷贝
- 相机采集使用环形缓冲区

3. 异步处理

```
// 相机采集与 AI 推理并行
await Task.WhenAll(
    _cameraSource.CaptureAsync(),
    _inferenceService.RunAsync(previousImage)
);
```

扩展性设计

添加新相机品牌

```
// 1. 实现 ICameraProvider 接口
public class BaslerCameraProvider : ICameraProvider
{
    public IEnumerable<CameraInfo> EnumerateDevices() { }
    public ICameraDevice Connect(CameraInfo info) { }
}

// 2. 注册到 CameraManager
CameraManager.RegisterProvider(new BaslerCameraProvider());
```

添加新 AI 任务

```
// 1. 实现推理器
public class OnnxOcrRunner : IOnnxRunner { }

// 2. 实现学习策略（如需增量学习）
public class OcrLearningStrategy : ILearningStrategy { }

// 3. 更新 registry.yaml 配置
```

测试策略

单元测试

```
[Fact]
public void VersionManager_CreateVersion_ShouldIncrementVersion()
{
    var manager = new VersionManager(testPath);
    var v1 = manager.CreateVersion("model.onnx");
    var v2 = manager.CreateVersion("model.onnx");
    Assert.Equal(v1.VersionNumber + 1, v2.VersionNumber);
}
```

属性测试 (Property-Based Testing)

```
[Property]
public Property Rollback_ShouldRestoreExactState()
{
    return Prop.ForAll<ModelVersion>(version =>
    {
        var backup = manager.Backup(version);
        manager.Rollback(backup);
        return manager.Current.Equals(version);
```

```
    });
}
```