Isaac Hung

Code details

27/11/2022

CODE CORE

App structure

- Tabs

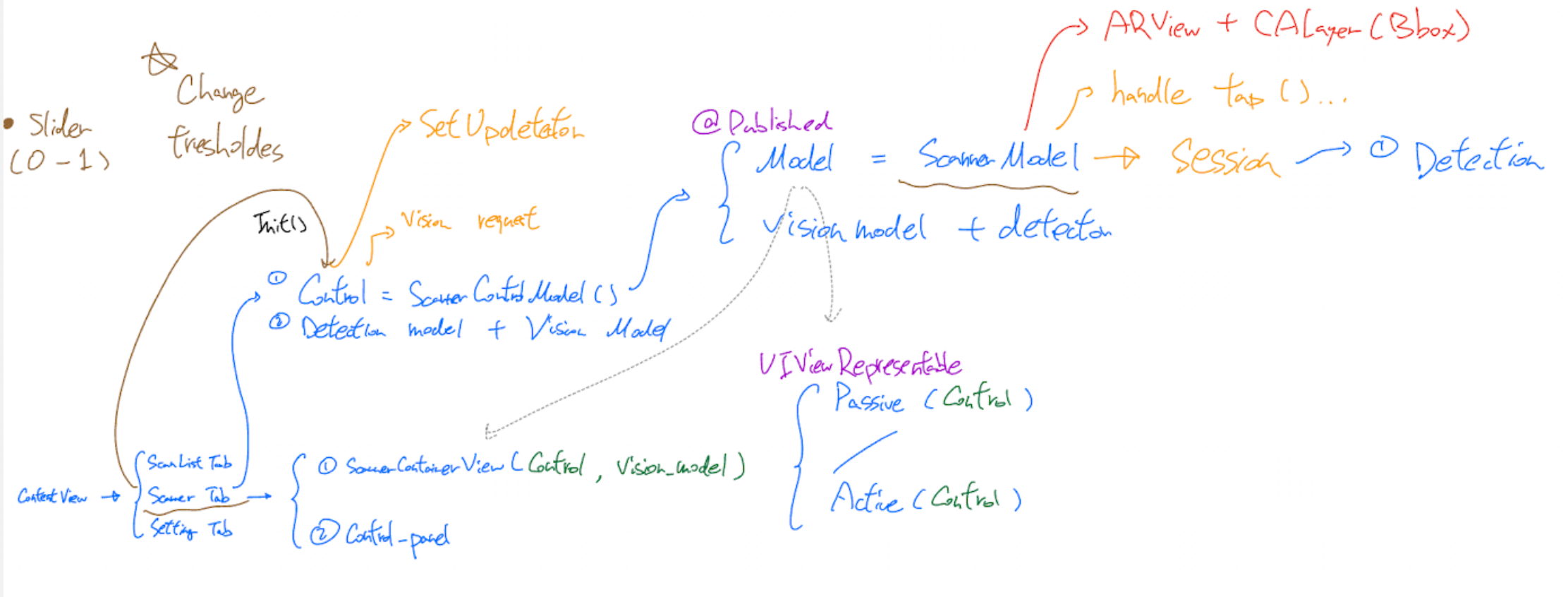
AR session

- Set up
- Marker
- Anchor and model
- Relocalization

AL model

- Set up
- BBox drawing layer

App structure - Tabs



- *** Everything is contolled by ScannerControlModel

AR session – set up

AR session

- In "ScannerModel.swift – func onViewAppear"

```swift
func onViewAppear(arView: ARView) {
    let surveyLines = DrawableContainer()
    let drawView = DrawOverlay(frame: arView.frame, toDraw: surveyLines)

    self.arView = arView
    self.drawView = drawView
    self.surveyLines = surveyLines
    self.savedAnchors.clear()

    setupARView(arView: arView)
    setupDrawView(drawView: drawView, arView: arView)
    arView.scene.subscribe(
        to: SceneEvents.Update.self
    ) {
        [weak self] in self?.updateScene(on: $0)
    }
    .store(in: &cancelBag)

    setupScanConfig()

    self.startScan()
    self.setupDetectionLayers()
      self.setupSCNLayers()
}
```

```swift
private func setupARView(arView: ARView) {
    #if !targetEnvironment(simulator)
    arView.automaticallyConfigureSession = false
    arView.debugOptions = .showSceneUnderstanding
    arView.renderOptions = [
        .disablePersonOcclusion,
        .disableMotionBlur,
        .disableDepthOfField,
        .disableHDR,
        .disableCameraGrain,
        .disableFaceMesh,
        .disableAREnvironmentLighting,
    ]
    #endif
}
```

```swift
private func setupScanConfig() {
    scanConfiguration = ARWorldTrackingConfiguration()
    scanConfiguration?.planeDetection = [.vertical, .horizontal]
    scanConfiguration!.sceneReconstruction = .meshWithClassification
    scanConfiguration!.environmentTexturing = .none
    //👆If automatic, manual -> Anchor will reflect the surrounding environment. But consume too much, then crash
    scanConfiguration!.worldAlignment = .gravityAndHeading
    scanConfiguration!.frameSemantics = .sceneDepth
}
```

```swift
arView.environment.sceneUnderstanding.options = [ .collision ]
arView.session.delegate = self
arView.session.run(
    scanConfiguration,
    options: [self.clearingOptions, .resetTracking]
)
```

AR session – set up

- Use ARView for environment reconstruction
  - Not ARSCNView!!!

- Use ARWorldTrackingConfiguration() to run the arsession

AR session

AR session – Marker

AR session

- In "ScannerModel.swift"
  - 1. func handleTap_for_yolo_4_pts ~~ 2. handleTap_auto_mark_4_pts

```
@objc func handleTap_auto_mark_4_pts(_ bboxs: [CGPoint], defect_name : String) {
```

  - 👆 Input: 4 x 2D pixel coordinates

```
let raycast_pt1    = arView.raycast( from: pt1, allowing: .estimatedPlane , alignment: .any ).first,
let raycast_pt2    = arView.raycast( from: pt2, allowing: .estimatedPlane, alignment: .any ).first,
let raycast_pt3    = arView.raycast( from: pt3, allowing: .estimatedPlane, alignment: .any ).first,
let raycast_pt4    = arView.raycast( from: pt4, allowing: .estimatedPlane, alignment: .any ).first,
let raycast_center = arView.raycast( from: center, allowing: .estimatedPlane, alignment: .any).first,
```

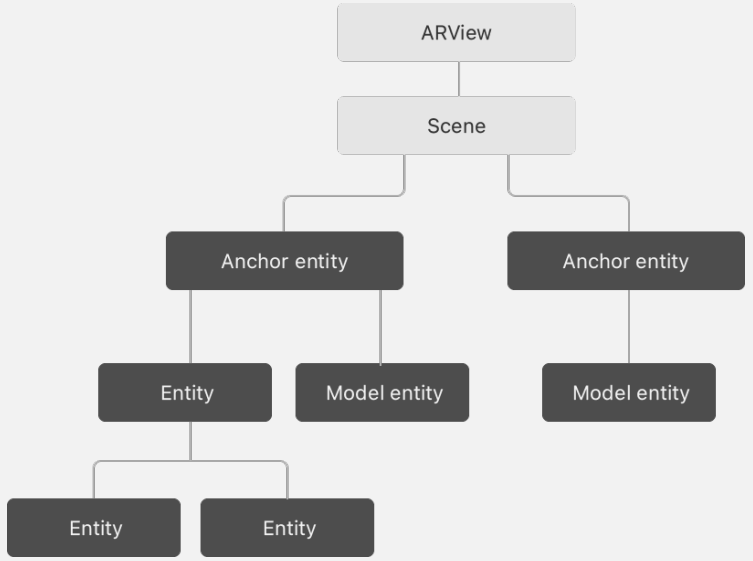
  - 👆From 2D pixel coordinates to 3D world coordinates

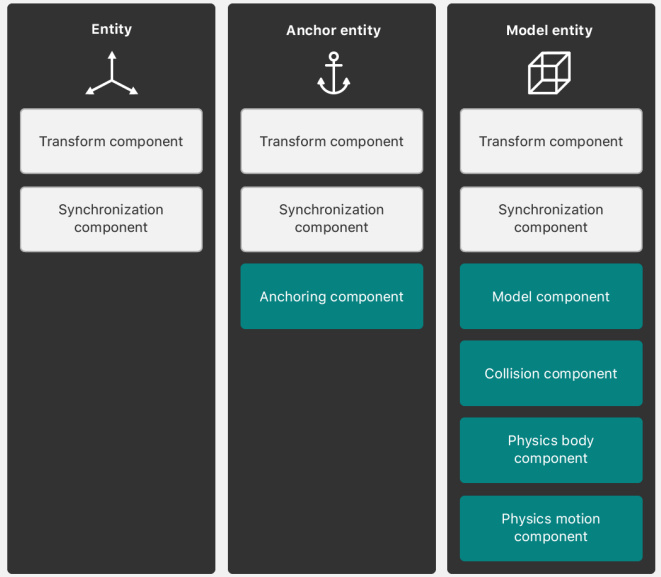
  - Add AnchorEntity -> Add ModelEntity to anchor -> Add AnchorEntity to ARsession

AR session – Anchor and model

AR session

- For all modelentity, must be attached to an Anchor entity
  - Position of modelentity = Relative position relative to its parent (Anchor entity)
  - Position of modelentity default as (0, 0, 0)
- https://developer.apple.com/documentation/realitykit/entity

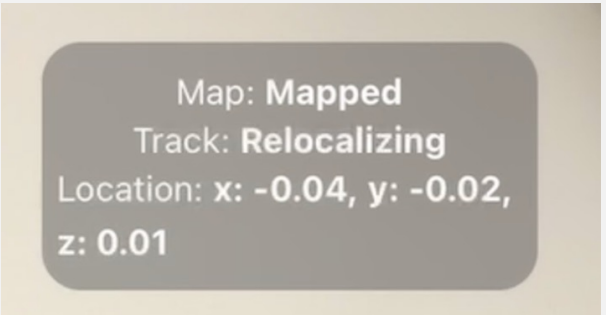




AR session – Relocalization

AR session

- Previous map save as “xxx_map.arexperience”
- If save map exsits, in func startScan()
  - Set initial World Map




```
if self.control!.reloc_map_url != FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)[0]{

    var mapDataFromFile: Data? {
        return try? Data(contentsOf: self.control!.reloc_map_url)
    }

    let worldMap: ARWorldMap = {
        guard let data = mapDataFromFile
            else { fatalError("Map data should already be verified to exist before Load button is enabled.") }
        do {
            guard let worldMap = try NSKeyedUnarchiver.unarchivedObject(ofClass: ARWorldMap.self, from: data)
                else { fatalError("No ARWorldMap in archive.") }
            return worldMap
        } catch {
            fatalError("Can't unarchive ARWorldMap from file data: \(error)")
        }
    }()

    scanConfiguration.initialWorldMap = worldMap
    print("##Isaac_debug \(self.control!.reloc_map_url)")

    control!.Recolized = false

}
```

AR session – Relocalization

AR session

- In "ScannerModel.swift - func session(_ session: ARSession, didAdd anchors: [ARAnchor]) "

1. Once relocated -> All the ARAnchor will be restored
2. Catch all the ARAnchor name with "reloc"
3. Add all the station back to the session

```
func session(_ session: ARSession, didAdd anchors: [ARAnchor]) {

    savedAnchors.update(anchors.compactMap { $0 as? ARMeshAnchor })

    if self.startSnapshot == nil { self.startSnapshot = SnapshotAnchor( capturing: session, suffix: "start" ) }

    var anchors_list: [ARAnchor] = []

    if !(control1.Recolized){
        anchors.forEach { anchor in
            guard let anchor_name = anchor.name else { return }
            if anchor_name.contains("reloc"){ anchors_list.append(anchor) }
        }

        anchors_list.sort { $0.name! < $1.name! }
          var BBox_side: Entity! = BBox_anchor.findEntity(named: "bbox_side")
          var BBox_side_model: ModelEntity! = BBox_side.children.first as? ModelEntity


        for i in stride(from: 0, through: anchors_list.count-1, by: 6) { //6 anchor(pts) per defect

            let mesh = MeshResource.generateSphere(radius: 0.01)

            let materials = [SimpleMaterial(color: .red, isMetallic: false)]
            let modelEntity = ModelEntity(mesh: mesh, materials: materials)

            guard let arView = self.arView,
                  let surveyLines = self.surveyLines else { return }

            let temp_station = Array(self.surveyStations)
            let MarkerID: Int = Int(temp_station.count) / 6 + 1
            let defect_name = String(anchors_list[i].name!.split(separator: "_")[1])


            let cam_ancr = anchors_list[i+0]
            let cen_ancr = anchors_list[i+1]
            let pt1_ancr = anchors_list[i+2]
            let pt2_ancr = anchors_list[i+3]
            let pt3_ancr = anchors_list[i+4]
            let pt4_ancr = anchors_list[i+5]

            print("cam_ancr.transform: ", cam_ancr.transform)
            print("cen_ancr.transform: ", cen_ancr.transform)
            print("pt1_ancr.transform: ", pt1_ancr.transform)
            print("pt2_ancr.transform: ", pt2_ancr.transform)
            print("pt3_ancr.transform: ", pt3_ancr.transform)
            print("pt4_ancr.transform: ", pt4_ancr.transform)

            let station_pt1 = SurveyStationEntity(worldTransform: pt1_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_pt1", RayQuery: nil,
                ARscene: arView.scene)
            let station_pt2 = SurveyStationEntity(worldTransform: pt2_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_pt2", RayQuery: nil,
                ARscene: arView.scene)
            let station_pt3 = SurveyStationEntity(worldTransform: pt3_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_pt3", RayQuery: nil,
                ARscene: arView.scene)
            let station_pt4 = SurveyStationEntity(worldTransform: pt4_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_pt4", RayQuery: nil,
                ARscene: arView.scene)
            let station_center = SurveyStationEntity(worldTransform: cen_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_Center", RayQuery:
                nil, ARscene: arView.scene)
            let station_camera = SurveyStationEntity(worldTransform: cam_ancr.transform, bbox_loc: "\(MarkerID)_\(defect_name)_Camera", RayQuery:
                nil, ARscene: arView.scene)

            let cam_ancr_entity = AnchorEntity(world: cam_ancr.transform)
            let cen_ancr_entity = AnchorEntity(world: cen_ancr.transform)
            let pt1_ancr_entity = AnchorEntity(world: pt1_ancr.transform)
            let pt2_ancr_entity = AnchorEntity(world: pt2_ancr.transform)
            let pt3_ancr_entity = AnchorEntity(world: pt3_ancr.transform)
            let pt4_ancr_entity = AnchorEntity(world: pt4_ancr.transform)

            station_center.removeFromParent(preservingWorldTransform: false)
            station_camera.removeFromParent(preservingWorldTransform: false)
            station_pt1.removeFromParent(preservingWorldTransform: false)
            station_pt2.removeFromParent(preservingWorldTransform: false)
            station_pt3.removeFromParent(preservingWorldTransform: false)
            station_pt4.removeFromParent(preservingWorldTransform: false)


            cam_ancr_entity.addChild(station_camera)
            cen_ancr_entity.addChild(station_center)
            pt1_ancr_entity.addChild(station_pt1)
            pt2_ancr_entity.addChild(station_pt2)
            pt3_ancr_entity.addChild(station_pt3)
            pt4_ancr_entity.addChild(station_pt4)


            arView.scene.addAnchor(pt1_ancr_entity)
            arView.scene.addAnchor(pt2_ancr_entity)
            arView.scene.addAnchor(pt3_ancr_entity)
            arView.scene.addAnchor(pt4_ancr_entity)
            arView.scene.addAnchor(cam_ancr_entity)
            arView.scene.addAnchor(cen_ancr_entity)
```

AL model - Set up

- In “ScannerControlModel.swift” -> Set Up

```
self.DetectionModel = {
    do {
        let configuration = MLModelConfiguration()
        return try yolov7x_original_with_NMS_IOU_Thrd_05_Conf_Thrd_04(configuration: configuration)
    } catch let error {
        fatalError(error.localizedDescription)
    }
}()

self.visionModel = try! VNCoreMLModel(for: DetectionModel.model)
```

- Live tuning iou & conf in “ScannerModel.swift”

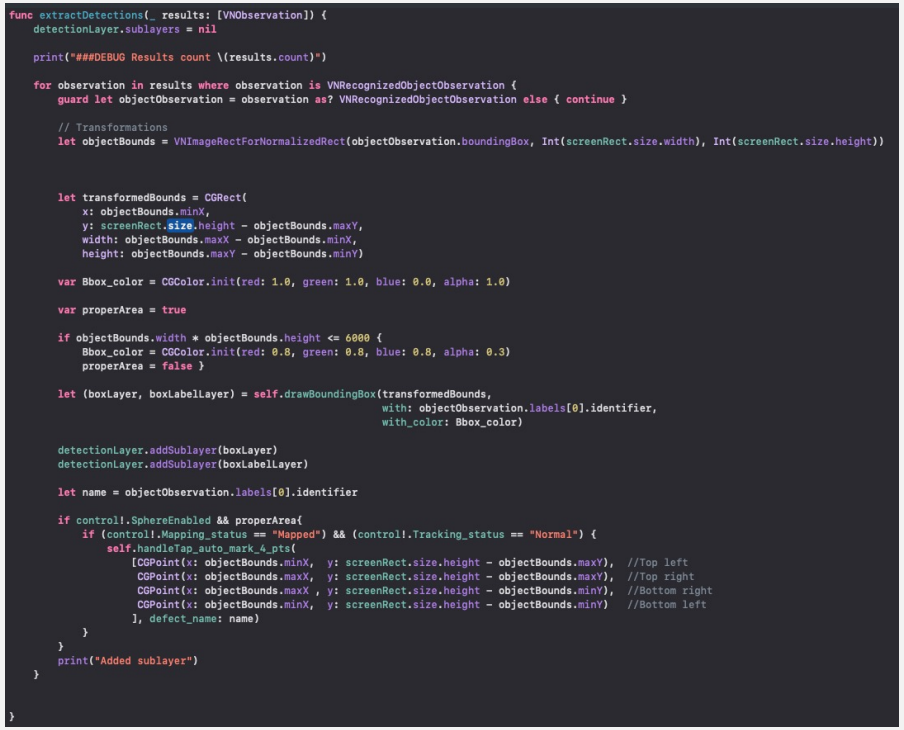
```
class ThresholdProvider: MLFeatureProvider {
    open var values = [
        "iouThreshold": MLFeatureValue(double: UserDefaults.standard.double(forKey: "iouThreshold")),
        "confidenceThreshold": MLFeatureValue(double: UserDefaults.standard.double(forKey: "confidenceThreshold"))
        ]

    var featureNames: Set<String> {
        return Set(values.keys)
    }

    func featureValue(for featureName: String) -> MLFeatureValue? {
        return values[featureName]
    }
}
```

AL model - BBox drawing layer

- In “ScannerModel.swift – func setupDetectionLayers()” -> Set Up

```swift
func setupDetectionLayers() {
    detectionLayer = CALayer()
    detectionLayer.frame = CGRect(x: 0, y: 0, width: screenRect.size.width, height: screenRect.size.height)
    self.arView!.layer.addSublayer(detectionLayer)
}
```

- In “ScannerModel.swift – func extractDetections()”
  - Draw Bbox and label when detected something



```swift
func extractDetections(_ results: [VNObservation]) {
    detectionLayer.sublayers = nil

    print("###DEBUG Results count \(results.count)")

    for observation in results where observation is VNRecognizedObjectObservation {
        guard let objectObservation = observation as? VNRecognizedObjectObservation else { continue }

        // Transformations
        let objectBounds = VNImageRectForNormalizedRect(objectObservation.boundingBox, Int(screenRect.size.width), Int(screenRect.size.height))



        let transformedBounds = CGRect(
            x: objectBounds.minX,
            y: screenRect.size.height - objectBounds.maxY,
            width: objectBounds.maxX - objectBounds.minX,
            height: objectBounds.maxY - objectBounds.minY)

        var Bbox_color = CGColor.init(red: 1.0, green: 1.0, blue: 0.0, alpha: 1.0)

        var properArea = true

        if objectBounds.width * objectBounds.height <= 6000 {
            Bbox_color = CGColor.init(red: 0.8, green: 0.8, blue: 0.8, alpha: 0.3)
            properArea = false }

        let (boxLayer, boxLabelLayer) = self.drawBoundingBox(transformedBounds,
                                                             with: objectObservation.labels[0].identifier,
                                                             with_color: Bbox_color)


        detectionLayer.addSublayer(boxLayer)
        detectionLayer.addSublayer(boxLabelLayer)

        let name = objectObservation.labels[0].identifier

        if control!.SphereEnabled && properArea{
            if (control!.Mapping_status == "Mapped") && (control!.Tracking_status == "Normal") {
                self.handleTap_auto_mark_4_pts(
                    [CGPoint(x: objectBounds.minX,  y: screenRect.size.height - objectBounds.maxY),  //Top left
                     CGPoint(x: objectBounds.maxX,  y: screenRect.size.height - objectBounds.maxY),  //Top right
                     CGPoint(x: objectBounds.maxX , y: screenRect.size.height - objectBounds.minY),  //Bottom right
                     CGPoint(x: objectBounds.minX,  y: screenRect.size.height - objectBounds.minY)   //Bottom left
                    ], defect_name: name)
            }
        }
        print("Added sublayer")
    }

}
```

AI Model