Add to proguard-rules.pro

ArcGIS keep rules (minimal)

-keep class com.esri.** { *; }

-dontwarn com.esri.**

-dontwarn okhttp3.**

-dontwarn okio.**

// REMOVE if not needed

implementation(libs.androidx.ui.tooling.preview)

Add this inside defaultConfig:

resConfigs("en")

Downgrade Compose BOM slightly:

composeBom = "2024.06.00"

✅ Stable
✅ Smaller
✅ Works perfectly with Kotlin 2.0

📉 Saves ~4–8 MB

❌ **You DON'T need these in production:**

androidx-ui-tooling

androidx-ui-test-manifest

androidx-ui-test-junit4

You already commented them in Gradle — GOOD.

But also remove them from version catalog to avoid accidental pulls.

🧠 **5. Kotlin 2.0 + Compose Plugin**

You're using:

kotlin = "2.0.21"

kotlin-compose = "2.0.21"

This is fine BUT:

⚠️ **Kotlin 2.0 currently generates slightly larger bytecode**

(compared to 1.9.24)

If size is more important than language features:

🔽 **Optional downgrade:**

kotlin = "1.9.25"

kotlin-compose = "1.9.25"

📉 Saves ~3–5 MB

6. Critical: Remove Debug Metadata

Add this to **gradle.properties**:

android.enableR8.fullMode=true

kotlin.compiler.execution.strategy=in-process

This allows **full shrinking & desugaring**.

Add to proguard-rules.pro:

ArcGIS core

-keep class com.esri.** { *; }

-dontwarn com.esri.**

Kotlin metadata reduction

-dontwarn kotlin.Metadata

-dontwarn kotlinx.**

Compose optimization

-keepclassmembers class * {

@androidx.compose.runtime.Composable <methods>;

}

2. BIGGEST SIZE CONTRIBUTORS (Confirmed)

🚨 1️⃣ **ArcGIS Toolkit (VERY HEAVY)**

import com.arcgismaps.toolkit.geoviewcompose.*

This adds:

- MapView Compose wrapper
- UI overlays
- Gesture handling
- Lifecycle adapters

📦 Adds **~20–30 MB**

➡️ **If you can live without toolkit**, you should remove it.

🚨 2️⃣ **Accompanist Permissions**

com.google.accompanist.permissions.*

Not huge (~1–2 MB), but **no longer necessary** on newer Android.

You can replace with:

ActivityResultContracts.RequestPermission

🚨 3️⃣ **Material3 + Navigation Compose**

These are okay, but you are importing more than needed.

✅ 3. CLEAN MINIMAL DEPENDENCY SET (RECOMMENDED)

✅ KEEP THESE

```
// Core
implementation(libs.androidx.core.ktx)
implementation(libs.androidx.lifecycle.runtime.ktx)

// Compose
implementation(platform(libs.androidx.compose.bom))
implementation(libs.androidx.ui)
implementation(libs.androidx.material3)

// Navigation
implementation("androidx.navigation:navigation-compose:2.8.0")

// ArcGIS core runtime ONLY
implementation(libs.arcgis.maps.kotlin)
```

❌ REMOVE THESE (IMPORTANT)

```
// Remove these
implementation("com.esri:arcgis-maps-kotlin-toolkit-geoview-compose")
implementation("com.google.accompanist:accompanist-permissions")
```

⚠️ 4. Replace Toolkit MapView (IMPORTANT)

You currently use:

```
MapView(
    modifier = Modifier.fillMaxSize(),
    map = map,
    mapViewProxy = mapViewProxy
```

)

Instead, use native MapView from ArcGIS SDK:

```
AndroidView(
    factory = { context ->
        MapView(context).apply {
            this.map = arcGISMap
        }
    },
    modifier = Modifier.fillMaxSize()
)
```

✓ Same functionality
✓ No toolkit dependency
✓ Smaller APK

⚠️ 5. Replace Accompanist Permissions

Instead of:

```
rememberPermissionState(...)
```

Use:

```
val launcher = rememberLauncherForActivityResult(
    ActivityResultContracts.RequestPermission()
) { isGranted ->
    // handle result
}
```

🧠 6. Compose Imports Cleanup (IMPORTANT)

You have duplicates and unused imports like:

❌ Repeated:

```
import androidx.compose.runtime.*
```

import androidx.compose.material3.*

Clean imports = faster build + smaller bytecode.

📦 7. FINAL DEPENDENCY LIST (CLEAN)

```
dependencies {

    // Core
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)

    // Compose
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.material3)

    // Navigation
    implementation("androidx.navigation:navigation-compose:2.8.0")

    // ArcGIS (core only)
    implementation(libs.arcgis.maps.kotlin)
}
```

FINAL OPTIMIZED build.gradle (Module)

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
```

```
    alias(libs.plugins.kotlin.compose)
    kotlin("plugin.serialization") version "2.0.0"
}

android {
    namespace = "com.example.nmsapp"
    compileSdk = 36

    defaultConfig {
        applicationId = "com.example.nmsapp"
        minSdk = 28
        targetSdk = 33
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"

        vectorDrawables {
            useSupportLibrary = true
        }

        // Reduce APK size
        resConfigs("en")
    }

    buildTypes {
        release {
            isMinifyEnabled = true
```

```
            isShrinkResources = true

            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )

            signingConfig = signingConfigs.getByName("debug")

            ndk {
                debugSymbolLevel = "NONE"
            }
        }
    }

    compileOptions {
        sourceCompatibility = JavaVersion.VERSION_11
        targetCompatibility = JavaVersion.VERSION_11
    }

    kotlinOptions {
        jvmTarget = "11"
    }

    buildFeatures {
        compose = true
    }
```

```
    packaging {
        resources {
            excludes += "/META-INF/{AL2.0,LGPL2.1}"
        }
        jniLibs {
            useLegacyPackaging = false
        }
    }

    // ✅ ONLY ARM64 (REAL DEVICES)
    splits {
        abi {
            isEnable = true
            reset()
            include("arm64-v8a")
            isUniversalApk = false
        }
    }
}

dependencies {

    // Core Android
    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)

    // Compose
```

```
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.material3)

    // ArcGIS Runtime (REQUIRED)
    implementation(libs.arcgis.maps.kotlin)

    // Navigation
    implementation("androidx.navigation:navigation-compose:2.8.0-beta06")

    // Permissions
    implementation("com.google.accompanist:accompanist-permissions:0.34.0")

    // Tests (kept minimal)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
}
```

✅ **ALSO ADD THIS (IMPORTANT)**

gradle.properties

```
android.enableR8.fullMode=true
```

✅ **proguard-rules.pro (REQUIRED)**

```
# ArcGIS Runtime
-keep class com.esri.** { *; }
-dontwarn com.esri.**
```

```
# Kotlin metadata
-dontwarn kotlin.Metadata

# Compose
-keepclassmembers class * {
    @androidx.compose.runtime.Composable <methods>;
}
```

CORE CONCEPTS (IMPORTANT)

ArcGIS uses:

Concept	Use
ArcGISMap	Main map
Basemap	Base map (imagery, topo, streets)
GraphicsOverlay	For points, lines, polygons
Graphic	Single shape
GeometryEngine	Distance / area calculations
FeatureLayer	Vector layers
ServiceFeatureTable	Data source
MapView	View rendering

🧩 **STEP 1: Create Base Map**

```
val arcGISMap = remember {
    ArcGISMap(BasemapStyle.ArcGISTopographic)
}
```

You can change base map:

BasemapStyle.ArcGISImagery

BasemapStyle.ArcGISStreets

🧩 **STEP 2: Setup MapView (Compose-friendly)**

```
@Composable
fun MapScreen() {
    val context = LocalContext.current

    val mapView = remember {
        MapView(context).apply {
            map = ArcGISMap(BasemapStyle.ArcGISTopographic)
        }
    }

    AndroidView(
        factory = { mapView },
        modifier = Modifier.fillMaxSize()
    )
}
```

🧩 **STEP 3: Load Vector Layer (GeoJSON / Feature Layer)**

Option A — GeoJSON (Offline Friendly)

```
val geoJsonSource = GeoJsonLayer(
    GeoJsonSource(File(context.filesDir, "roads.geojson").path)
)

map.operationalLayers.add(geoJsonSource)
```

Option B — Feature Service (Online)

```
val featureTable = ServiceFeatureTable(
    "https://services.arcgis.com/.../FeatureServer/0"
)

val featureLayer = FeatureLayer(featureTable)
map.operationalLayers.add(featureLayer)
```

🧩 STEP 4: Toggle Vector Layer ON/OFF

```
var showLayer by remember { mutableStateOf(true) }

Button(onClick = {
    showLayer = !showLayer
    featureLayer.isVisible = showLayer
}) {
    Text(if (showLayer) "Hide Layer" else "Show Layer")
}
```

🧩 STEP 5: Add Points via Long Press

Create GraphicsOverlay

```
val graphicsOverlay = GraphicsOverlay()
mapView.graphicsOverlays.add(graphicsOverlay)
```

Capture Long Press

```
mapView.onTouchListener = object : DefaultMapViewOnTouchListener(context, mapView) {
    override fun onLongPress(e: MotionEvent) {
        val point = mapView.screenToLocation(android.graphics.Point(e.x.toInt(), e.y.toInt()))

        val marker = Graphic(
```

```
            point,
            SimpleMarkerSymbol(SimpleMarkerSymbol.Style.CIRCLE, Color.RED, 10f)
        )

        graphicsOverlay.graphics.add(marker)
    }
}
```

🧩 STEP 6: Draw Line / Polygon Between Points

Store Points

```
val points = mutableListOf<Point>()
```

Add point:

```
points.add(point)
```

Draw Line

```
val polyline = Polyline(points)
val lineSymbol = SimpleLineSymbol(SimpleLineSymbol.Style.SOLID, Color.BLUE, 3f)

graphicsOverlay.graphics.add(Graphic(polyline, lineSymbol))
```

Draw Polygon

```
val polygon = Polygon(points)
val fill = SimpleFillSymbol(
    SimpleFillSymbol.Style.SOLID,
    Color(0x5500FF00),
    SimpleLineSymbol(SimpleLineSymbol.Style.SOLID, Color.GREEN, 2f)
)

graphicsOverlay.graphics.add(Graphic(polygon, fill))
```

🧩 **STEP 7: Calculate Distance Between Points**

```
val distance = GeometryEngine.distanceGeodetic(
    points[0],
    points[1],
    LinearUnit(LinearUnitId.METERS),
    AngularUnit(AngularUnitId.DEGREES),
    GeodeticCurveType.GEODESIC
)

val meters = distance.distance
```

For multiple points:

```
val totalDistance = GeometryEngine.lengthGeodetic(
    Polyline(points),
    LinearUnit(LinearUnitId.METERS),
    GeodeticCurveType.GEODESIC
)
```

🧩 **STEP 8: Highlight Selected Area**

Use a semi-transparent polygon:

```
val highlightSymbol = SimpleFillSymbol(
    SimpleFillSymbol.Style.SOLID,
    Color(0x5500FF00),
    SimpleLineSymbol(SimpleLineSymbol.Style.DASH, Color.GREEN, 2f)
)
```

🎯 **FINAL STRUCTURE (Clean Architecture)**

```
ui/
  ├– MapScreen.kt
```

├– MapControls.kt

└– MapState.kt

map/

├– MapManager.kt

├– GeometryUtils.kt

🚀 **RESULT**

✓ Offline + Online maps
✓ Vector overlays
✓ Toggle visibility
✓ Draw shapes
✓ Distance measurement
✓ Efficient + optimized
✓ No heavy toolkit