Improved Prompt for Creating Android XR SDK Reference Artifact

Original Goal

Create a technical reference that teaches Claude (or other reasoning models) about Android XR SDK Developer Preview 3 APIs to prevent hallucination when architecting applications.

Enhanced Prompt

I need you to create a comprehensive technical reference artifact for Android XR SDK Developer Preview 3 that will serve as a knowledge base for architectural decisions AND code generation. This artifact must be structured to minimize API hallucination when a reasoning model uses it.

Source Material: Perplexity research report (provided in documents)

Critical Requirements:

1. API Precision Layer

For each API or library mentioned, include:

- **Exact package/import paths** (e.g., `androidx.xr.compose.glimmer.*`)
- **Concrete class names** (e.g., `ProjectedActivity`, `SurfaceEntity`)
- **Key method signatures** with parameter types and return types
- **Code snippets** showing actual usage patterns (even if minimal/pseudocode based on documentation patterns)
- **Version/availability notes** (which DP version introduced this)

2. Hierarchical Organization

Structure information in layers:

- **Layer 1: Conceptual Overview** - What the API does and why it exists
- **Layer 2: Technical Specification** - Exact classes, methods, types
- **Layer 3: Implementation Patterns** - How to use it in practice
- **Layer 4: Constraints & Gotchas** - What won't work, common mistakes

3. Code-Generative Format

For each major API category, provide:

- **Minimal working example** structure (even if some details are unknown)
- **Required imports** that can be directly copied
- **Initialization patterns** showing typical setup
- **Lifecycle integration points** (where in Activity/Fragment lifecycle things happen)
- **Clear markers** for "known exact API" vs "inferred pattern" vs "needs documentation lookup"

4. Decision Trees with Technical Details

When presenting "when to use X vs Y" decisions:

- Include the **specific APIs/classes** involved in each path
- Show **code skeleton differences** between approaches
- Indicate **dependency requirements** (Gradle dependencies, permissions, etc.)
- List **performance/capability trade-offs** with measurable criteria

5. Anti-Hallucination Safeguards

Explicitly include:

- **"Unknown/Not Specified" sections** - clearly mark what information is NOT in the source material
- **Documentation references** - link to specific doc pages where full API details exist
- **Confidence indicators** - mark whether information is:
  - ✓ Directly from source (quote/cite)
  - ~ Inferred from patterns (logical extrapolation)
  - ? Requires verification (guess that needs checking)

6. Quick Reference Tables

Create tables that map:

- **Feature → Exact Class/Library → Package Name**
- **Use Case → Required Permissions → API Level**
- **Device Type → Supported APIs → Initialization Pattern**
- **Common Task → Code Template Reference**

7. Integration Patterns

For cross-cutting concerns, show:

- **How Jetpack Projected integrates with Compose Glimmer** (code structure)
- **How ARCore APIs connect to SceneCore rendering** (data flow)
- **How to handle multi-device scenarios** (runtime detection code)
- **Error handling patterns** for each API category

8. Search Optimization

Structure content so a reasoning model can quickly locate:

- "How do I [specific task]?" → Direct to code template
- "What's the class for [concept]?" → Direct to API specification
- "Can I use [feature] on [device]?" → Direct to compatibility matrix
- "What imports do I need for [functionality]?" → Direct to import list

9. Completeness Indicators

For each API section, explicitly state:

- ✓ **Complete** - Source provides full API details
- ◐ **Partial** - Source provides concept but missing implementation details
- ✗ **Incomplete** - Source mentions existence but provides no details

10. Citable Extractions

When source material includes specific technical details:

- **Extract verbatim** any code snippets, class names, method names
- **Preserve exact terminology** from documentation
- **Quote specific numbers** (e.g., "68 blendshapes", "version 36.4.3")
- **Cite source** for each technical claim

Output Format Requirements:

1. **Use Markdown** with clear heading hierarchy
2. **Code blocks** for all code examples with language tags

3. **Tables** for matrices and mappings
4. **Callout sections** for warnings, unknowns, and gotchas
5. **Index section** at the end for quick navigation

Success Criteria:

When a reasoning model uses this artifact to generate code:

- It should **never invent** class names, method names, or package paths
- It should **clearly indicate** when it lacks specific implementation details
- It should **reference the artifact section** that informed its response
- It should **suggest documentation lookup** for missing details rather than guessing

---

Using the Perplexity research report provided, create this enhanced technical reference artifact now.