Android Markdown 渲染库调研（面向本项目：Chat + Files）

Executive Summary

本项目同时包含 **Jetpack Compose（Chat）** 与 **View/RecyclerView（Files）** 两种 UI 形态，因此 Markdown 渲染方案要么选择“View 侧成熟库 + Compose 侧用 AndroidView 包一层”，要么选择“Compose 原生 Markdown 库”。综合成熟度、功能完备性和落地成本，**Markwon** 更适合当作第一期的默认渲染引擎；若后续希望完全 Compose 化，可再评估 `compose-richtext` 等方案。 [1][2]

Key Findings

- **Markwon（Noties）是 Android 上最成熟的 Markdown 渲染方案之一**：基于 `commonmark-java`，把 Markdown 渲染成 `TextView` 的 spans，并提供插件体系扩展（图片、表格、语法高亮等）。 [1][2]
- **Compose 原生 Markdown 渲染仍在快速演进**：`compose-richtext` 提供 Markdown/HTML 在 Compose 内渲染，但官方明确标注为实验性 API，升级/行为变化风险更高。 [3]
- **CommonMark 生态是“正确解析 Markdown”的基础**：CommonMark 规范定义了更一致的 Markdown 行为；`commonmark-java` 是其 Java 实现，常被上层渲染库复用。 [4][5]

Detailed Analysis

1) View 体系优先：Markwon（推荐作为本项目当前默认）

适配性

- Files 页签当前是 RecyclerView + Dialog（View 系统），Markwon 可以直接在 `TextView` 上渲染，无需迁移 UI。 [2]
- Chat 页签是 Compose，但可以在气泡内部用 `AndroidView` 包一层 `TextView`，复用同一套 Markwon 渲染能力（插件、链接、列表、引用、代码块等）。 [2]

能力与扩展

- Markwon 的核心思路是把 Markdown 解析后转成 spans 并设置到 `TextView` 上；并通过插件扩展图片、表格、语法高亮等。 [1]
- Markwon 明确基于 `commonmark-java`（CommonMark Java 实现），解析行为相对稳定。 [1][5]

风险点

- Markwon 的“载体”是 `TextView`：在 Compose 列表里大量使用 `AndroidView` 可能带来一定的测量/复用成本，需要在后续结合实际消息量做性能观察。
- 高级特性（图片、代码高亮、表格）需要额外插件与依赖，引入时要关注体积与加载时延。 [1]

2) Compose 原生方案：compose-richtext（备选）

优点

- 纯 Compose 渲染，不需要 `AndroidView` 嵌套；更容易跟 Material 3 的字体/颜色/布局体系一致。 [3]

风险点

- 文档明确标注该库的 Markdown/HTML 等能力属于 **实验性 API**（需要 `@OptIn(ExperimentalRichTextApi::class)`），意味着版本升级可能产生 breaking changes。 [3]

3) “只选解析器”：commonmark-java + 自研渲染

如果未来你希望实现“完全自定义的 Markdown 渲染风格”和更强的一致性，也可以只引入解析器（`commonmark-java`）并自研 renderer。但这会显著增加开发/测试成本，尤其是列表/嵌套/引用/代码块/链接等细节。 [5]

Areas of Consensus

- **CommonMark 规范/实现**能显著降低 Markdown 解析差异，减少“同一段 Markdown 在不同端渲染不同”的问题。 [4][5]
- 如果项目同时存在 Compose 与 View，**先落地成熟的 View 渲染**（再在 Compose 中桥接）通常是最快闭环路线。 [2]

Areas of Debate

- **第一期是否坚持 Compose 原生**：Compose 原生库更“纯”，但实验性/迭代快带来的维护成本可能更高。 [3]

- **流式增量渲染的体验**：边流式输出边渲染 Markdown 可能出现“未闭合代码块/列表”的中间态；可考虑在流式阶段先纯文本展示，完成后再切到 Markdown（取决于体验取舍）。

Sources

[1] Noties/Markwon GitHub README（权威：项目官方文档；说明其基于 commonmark-java、TextView spans、插件体系等）。https://github.com/noties/Markwon
[2] Markwon “Getting started” 文档（权威：项目官方文档；给出 TextView 集成方式与实现依赖 commonmark-java 的说明）。https://noties.io/Markwon/docs/v4/getting-started.html
[3] `compose-richtext` 官方文档（权威：项目官方文档；声明 Experimental API、功能范围）。https://halilozercan.github.io/compose-richtext/markdown/
[4] CommonMark 官方规范（权威：规范原文；定义 Markdown 语法与一致性目标）。https://spec.commonmark.org/
[5] `commonmark-java` GitHub（权威：实现官方仓库；说明其为 CommonMark Java 实现）。https://github.com/commonmark/commonmark-java

Gaps and Further Research

- 图片渲染与缓存：Markwon 需要选择图片插件（Coil/Glide）与缓存策略；Compose 原生库的图片能力也需要评估。
- 代码块体验：是否需要语法高亮与复制按钮（对工具 trace / agent 输出很常见）。
- 大消息列表性能：随着会话变长，`AndroidView` 方案在 Compose LazyColumn 中的性能上限需要基准测试。