@copilotkit/react-ui 与 AG-UI 协议完全指南（2025 年 12 月最新版）

一、什么是 AG-UI 协议?

AG-UI（Agent-User Interaction Protocol） 是 CopilotKit 团队于 2024 年底推出的开源、轻量级、基于事件流的「代理 ↔ 前端」通信协议，被称为「Agentic UI 的 HTTP」。

它彻底解决了传统 AI 聊天界面只能「纯文字」的痛点，让后端 Agent（LangGraph、CrewAI、LlamaIndex、OpenAI Assistants、ADK、AG2 等）可以：

- 实时流式输出中间思考过程
- 实时同步任意 JSON 状态到前端（State Sync）
- 动态生成 UI（表单、按钮、卡片等）
- 请求用户确认（Human-in-the-Loop）
- 直接操作前端 React state

AG-UI 协议官网：https://ag-ui.dev GitHub（独立协议仓库）：https://github.com/ag-ui-protocol/ag-ui-protocol

二、@copilotkit/react-ui 是 AG-UI 的「官方前端客户端」

@copilotkit/react-ui 从 v1.50 起已 100% 基于 AG-UI 协议构建，成为目前最成熟、最易用的 AG-UI 前端实现。

组件 / Hook	作用	AG-UI 事件支持情况
CopilotChat	完整聊天窗口	所有事件（文本、状态、工具、生成 UI）
CopilotPopup	右下角浮窗（ChatGPT 风格）	同上
CopilotSidebar	侧边栏助手（GitHub Copilot 风格）	同上
useAgent()	新一代核心 Hook，直接连接任意 AG-UI 后端	完整支持
useCoAgentStateRender	专门渲染 Agent 实时状态	STATE_DELTA 专用
useCopilotAction	前端动作（旧版）→ 逐步被 AG-UI 取代	兼容层

三、AG-UI 协议核心事件类型（2025 最新）

事件类型	方向	说明	前端典型渲染方式

TEXT_MESSAGE_CONTENT	Agent→UI	流式文本片段	打字机效果
STATE_DELTA	Agent→UI	JSON Patch 格式的状态更新	实时更新行程、表格、图表等
TOOL_CALL_START / END	Agent→UI	工具调用开始/结束	显示“正在搜索航班…”进度条
GENERATED_UI	Agent→UI	动态生成 React 组件（JSON Schema → Component）	自动弹出表单、按钮组
REQUEST_HUMAN_INPUT	Agent→UI	暂停等待用户输入	弹出确认框、输入框
USER_MESSAGE	UI→Agent	用户发送的消息	普通文本
ACTION_EXECUTION	UI→Agent	用户确认后执行动作	提交表单数据

四、完整实战示例：旅行规划 Agent（实时同步行程）

1. 前端（Next.js 15 + App Router）

tsx

```
// app/layout.tsx
import { CopilotKit } from "@copilotkit/react-core";

export default function RootLayout({ children }: { children: React.ReactNode
}) {
  return (
    <html lang="zh">
      <body>
        {/*         AG-UI         */}
        <CopilotKit url="/api/copilotkit">
          {children}
        </CopilotKit>
      </body>
    </html>
  );
}
```

tsx

```
// app/page.tsx
"use client";
import { useAgent } from "@copilotkit/react-core";
import { CopilotSidebar, CopilotChat } from "@copilotkit/react-ui";

export default function Home() {
```

```
  const {
    messages,
    append,
    agentState,        //
    isLoading,
    isExecuting,
  } = useAgent({
    agent: "travel-planner-v2",           //          Agent
    initialMessages: [{ role: "user", content: "       12  20-22           "
}]
  });

  //         Agent
  const outline = agentState?.outline as string[] | undefined;

  return (
    <div className="flex h-screen">
      <div className="flex-1 p-8">
        <h1 className="text-3xl font-bold mb-8">          </h1>

        {outline ? (
          <div className="space-y-4">
            {outline.map((day, i) => (
              <div key={i} className="bg-blue-50 p-6 rounded-xl border
border-blue-200">
                <h3 className="font-semibold text-lg">   {i+1}   </h3>
                <p className="text-gray-700">{day}</p>
              </div>
            ))}
          </div>
        ) : (
          <p className="text-gray-500">AI           …</p>
        )}
      </div>

      {/*          +            */}
      <CopilotSidebar>
        <CopilotChat
          labels={{
            title: "            AG-UI  ",
            initial: "                    ",
          }}
          //          Agent
          renderMessage={(message) => {
            if (message.role === "assistant" && message.content.includes("
        )) {
```

```
              return <span className="italic text-gray-500">{message.content}
⏳</span>;
            }
            return undefined;
          }}
        />
      </CopilotSidebar>
    </div>
  );
}
```

2. 后端示例（Python + FastAPI + LangGraph）

Python

```
# backend/main.py
from fastapi import FastAPI
from copilotkit.integrations.ag_ui import ag_ui_endpoint
from my_travel_agent import travel_planner_graph  #     LangGraph

app = FastAPI()

#            AG-UI
app.add_api_route(
    "/api/copilotkit",
    ag_ui_endpoint(travel_planner_graph),
    methods=["POST"]
)
```

Python

```
# my_travel_agent.py
async def planner_node(state):
    #              emit
    new_outline = ["Day1:                  ", "Day2:     +        "]

    copilotkit.emit_state_delta({"outline": new_outline})  # ← AG-UI

    #
    weather = await get_weather("    ")
    copilotkit.emit_message(f"              3       {weather}")

    return {"outline": new_outline, "weather": weather}
```

五、为什么 2025 年你应该用 @copilotkit/react-ui + AG-UI?

优势	传统方案（纯 OpenAI API）	@copilotkit/react-ui + AG-UI

实时显示 Agent 思考过程	不支持	原生支持
实时同步复杂状态	自己写 WebSocket + JSON Patch	开箱即用
动态生成 UI	几乎不可能	一句 GENERATED_UI 事件
Human-in-the-Loop	手动实现	协议级支持
多框架兼容	只能 React	前端任意框架，后端任意 Agent
开发速度	2～4 周	1～3 天

六、官方资源（全部 2025 年 12 月最新）

- 官网：https://www.copilotkit.ai
- AG-UI 协议官网：https://ag-ui.dev
- 文档（含完整 API）：https://docs.copilotkit.ai
- GitHub 主仓库（30k+ stars）：https://github.com/CopilotKit/CopilotKit
- AG-UI 协议仓库：https://github.com/ag-ui-protocol/ag-ui-protocol
- 30+ 开箱示例：https://github.com/CopilotKit/CopilotKit/tree/main/examples

如果你正在用 LangGraph、CrewAI、LlamaIndex、ADK 或任何自定义 Agent，强烈推荐直接上 @copilotkit/react-ui + AG-UI，几乎是 2025 年最省事的「Agentic 前端」方案。

AG-UI 协议高级特性详解（2025 年 12 月最新版，新增多个真实实例）

一、AG-UI 协议概述

AG-UI（Agent-User Interaction Protocol） 是 CopilotKit 主导的开源协议，已成为 2025 年 Agentic UI 的事实标准。它支持实时双向通信，已集成 Microsoft Agent Framework、AWS Strands、AG2、LangGraph、CrewAI、LlamaIndex 等主流框架。

- **最新里程碑**：CopilotKit v1.50 原生 AG-UI；useAgent Hook 发布；协议事件扩展至 17 种；生态下载量超千万。
- **核心价值**：标准化事件流（HTTP+SSE），实现状态同步、生成式 UI、人类在环、多 Agent 协调。

二、AG-UI 的 17 种核心事件类型（分类详解）

类别	主要事件类型	方向	高级用途	典型应用场景
消息与文本	TEXT_MESSAGE_CONTENT, MESSAGE_START/END	Agent → UI	流式渲染、部分消息优先显示	Agent 实时“思考”输出
工具调用	TOOL_CALL_START/END, TOOL_RESULT	双向	进度可视化、结果注入 UI	显示“正在查询股票数据…”
状态管理	STATE_DELTA, STATE_SNAPSHOT	双向	增量 Patch 更新大状态、可靠恢复	实时更新投资组合仪表盘

生成式 UI	GENERATED_UI, STRUCTURED_MESSAGE	Agent → UI	动态渲染表单/卡片/图表（兼容 A2UI）	Agent 自动生成天气卡片
人类在环	REQUEST_HUMAN_INPUT, ACTION_EXECUTION	双向	审批/修改/反馈循环	用户确认发送邮件草稿
生命周期与扩展	THREAD_START/END, CUSTOM_EVENT	双向	会话持久化、自定义事件转发	多 Agent 协作中断恢复

三、高级特性详解

1. 双向状态同步（Bi-directional State Synchronization）

- **机制**：使用 RFC 6902 JSON Patch 格式的 STATE_DELTA 事件，仅传输变化部分（delta），高效处理大状态（如 JSON 树、笔记本、仪表盘）。
- **快照恢复**：STATE_SNAPSHOT 用于重建基线，防止补丁应用失败。
- **高级应用**：支持复杂协作，如 Agent 更新计划，用户实时修改并反馈给 Agent，形成闭环。
- **优势**：低带宽、实时一致性，支持会话持久化（thread persistence）和可靠重连。

2. 人类在环（Human-in-the-Loop, HITL）协作

- **核心事件**：REQUEST_HUMAN_INPUT 暂停 Agent，等待用户响应；ACTION_EXECUTION 执行用户批准动作。
- **高级模式**：
  - 审批流程：Agent 生成邮件/代码，用户确认后执行。
  - 反馈循环：用户修改状态 delta，Agent 据此调整。
  - 中断支持：用户可随时取消长任务。
- **集成示例**：CopilotKit 的 renderAndWaitForResponse 支持自定义审批 UI（如复选框、输入框）。

3. 生成式 UI（Generative UI）与结构化消息

- **机制**：Agent 发送 GENERATED_UI 事件（JSON Schema），前端自动渲染组件（表单、表格、图表）。
- **与 A2UI 互补**：A2UI 定义 UI 小部件规范，AG-UI 提供运行时双向连接。
- **高级应用**：动态仪表盘、交互卡片，支持自定义渲染钩子。

4. 实时上下文丰富与前端工具集成

- **特性**：前端可注册 React state 作为上下文，Agent 实时读取；支持前端工具调用。
- **优势**：Agent 可直接操作应用数据（如更新 Todo 列表），实现真正“Agentic”体验。

5. 多 Agent 协调与线程管理

- **支持**：线程 ID（thread_id）确保会话隔离；与 A2A 协议结合，实现多 Agent 协作。
- **高级**：可靠重连、会话超时、类型安全。

6. 安全性、可观测性与治理

- **内置**：事件流作为审计轨迹，支持 RLiHF（隐式人类反馈）优化。
- **守栏**：防止提示注入、数据泄露；中间件支持审批/日志。

- **调试**：实时事件日志、会话回放（AG-UI Dojo 示例）。

四、AG-UI 高级特性真实实例（新增 6 个生产级案例）

以下实例均来自官方博客、GitHub 示例及社区集成（2025 年最新），涵盖不同框架和场景。

1. 股票投资组合 Agent（LangGraph + AG-UI）

- **场景**：用户输入股票列表，Agent 实时分析、市场数据查询、生成投资建议，并同步仪表盘。
- **关键特性**：STATE_DELTA 实时更新图表；TOOL_CALL_START 显示查询进度；GENERATED_UI 动态渲染股票卡片。
- **实现**：LangGraph 后端 emitIntermediateState；前端 useCoAgentStateRender 渲染。
- **资源**：完整教程 https://www.copilotkit.ai/blog/build-a-fullstack-stock-portfolio-agent-with-langgraph-and-ag-ui

2. 旅行规划多 Agent 系统（AG2 + FastAgency + AG-UI）

- **场景**：多 Agent 协作（航班、酒店、行程 Agent），实时生成个性化行程，用户可修改并反馈。
- **关键特性**：多 Agent 协调（A2A + AG-UI）；Human-in-the-Loop 用户审批酒店；STATE_SNAPSHOT 会话恢复。
- **实现**：AG2 工作流 + CopilotKit Runtime 中间件。
- **资源**：Starter Repo（含完整代码） https://docs.ag2.ai/latest/docs/blog/2025/05/07/AG2-Copilot-Integration/

3. 数据分析仪表盘 Agent（LangGraph + React Dashboard）

- **场景**：Agent 接收查询，规划图表、拉取数据、实时更新仪表盘，用户可迭代查询。
- **关键特性**：STATE_DELTA 流式更新图表数据；多轮交互闭环。
- **实现**：LangGraph planner 节点 emit_state_delta。
- **资源**：官方博客真实案例 https://www.copilotkit.ai/blog/ag-ui-protocol-bridging-agents-to-any-front-end

4. 邮件审批与发送 Agent（Human-in-the-Loop 经典）

- **场景**：Agent 根据上下文生成邮件草稿，弹出确认框，用户编辑/批准后发送。
- **关键特性**：REQUEST_HUMAN_INPUT 暂停；renderAndWaitForResponse 自定义审批 UI。
- **代码片段**（CopilotKit 示例）：

  tsx

```
useCopilotAction({
  name: "send_email",
  renderAndWaitForResponse: ({ args, respond }) => (
    <EmailConfirmation
      draft={args.email_draft}
      onApprove={() => respond({ approved: true })}
      onEdit={(edited) => respond({ approved: true, editedDraft: edited
})}
    />
```

```
  ),
});
```

- **资源**：GitHub CopilotKit README 示例

5. 餐厅推荐 & 预订 Agent（CrewAI + AG-UI）

- **场景**：用户描述偏好，CrewAI 多角色 Agent（研究员、预订员）协作，实时显示选项并预订。
- **关键特性**：TOOL_RESULT 注入卡片；Human-in-the-Loop 确认预订。
- **实现**：CrewAI workflow + AG-UI 端点。
- **资源**：完整仓库 https://github.com/CopilotKit 示例 / 博客 https://www.copilotkit.ai/blog/how-to-add-a-frontend-to-any-crewai-agent-using-ag-ui-protocol

6. 研究画布 Agent（LangGraph Canvas Research Demo）

- **场景**：Agent 进行深度研究，实时在画布上显示来源、摘要、关系图，用户可指导下一步。
- **关键特性**：中间状态流式（思考过程）；GENERATED_UI 渲染关系图。
- **实现**：LangGraph + CopilotSidebar。
- **资源**：在线 Demo https://www.copilotkit.ai/examples/canvas-research/

五、为什么选择 AG-UI?

（保持原有对比表）

六、资源汇总（2025 年 12 月最新）

- **AG-UI Dojo**（互动 Demo，10+ 框架场景）：https://docs.ag-ui.com/dojo
- **CopilotKit 示例页**：https://www.copilotkit.ai/examples
- **博客教程合集**：https://www.copilotkit.ai/blog （股票、旅行、CrewAI、Microsoft 等）
- **GitHub**：https://github.com/ag-ui-protocol/ag-ui （协议） & https://github.com/CopilotKit/CopilotKit （实现 + 示例）

这些实例覆盖了从简单 HITL 到复杂多 Agent 协作的全谱系，几乎所有主流框架都已原生支持。如果你想深入某个实例（比如股票 Agent 的完整代码），或需要我帮你搭建一个自定义模板，直接说场景，我可以一步步指导！🚀

将 @copilotkit/react-ui 渲染引擎集成到 Android、iOS 或 Linux 原生应用中的方案（2025 年 12 月最新）

@copilotkit/react-ui 是基于 **React** 的 Web 前端 UI 包（用于 CopilotChat、CopilotSidebar 等组件），它本身不是原生移动或桌面组件。因此，直接在纯原生 Android (Kotlin/Java)、iOS (Swift) 或 Linux 应用中运行 React UI 需要一些桥接方式。

好消息是：CopilotKit 的核心交互依赖 **AG-UI 协议**（Agent-User Interaction Protocol），而 AG-UI 在 2025 年 11 月已推出 **Kotlin Multiplatform SDK**，支持直接在 **Android、iOS 和 JVM** 原生环境中实现完整 Agent 连接，无需 WebView 或 JavaScript 桥接。这意味着你可以用原生 UI（Jetpack Compose、SwiftUI）构建界面，同时保留完整的实时状态同步、Human-in-the-Loop 等 Agentic 特性。

下面按平台分方案详解：

1. Android 和 iOS（移动原生应用）推荐方案：使用 AG-UI Kotlin SDK（原生、无 WebView）

- **最新进展**（2025 年 11 月）：社区贡献者 Mark Fogle 发布了 AG-UI 的 **Kotlin Multiplatform SDK**，支持 Android、iOS 和 JVM 一份代码跨平台。
- **优势**：
  - 原生性能：无 JS 桥接开销。
  - 完整 AG-UI 特性：实时消息流、状态 delta、工具调用、生成式 UI、Human-in-the-Loop。
  - 与你的 React UI 逻辑一致：后端 Agent（LangGraph 等）不变，前端用 Kotlin 实现相同交互。
- **实现步骤**：
  1. 在项目中添加 AG-UI Kotlin SDK（Gradle/Maven 依赖，详见官方博客）。
  2. 用 Jetpack Compose (Android) 或 SwiftUI (iOS) 构建聊天界面。
  3. 通过 SDK 连接你的 AG-UI 后端端点（/api/copilotkit）。
  4. 处理事件流：订阅 TEXT_MESSAGE_CONTENT 显示消息、STATE_DELTA 更新 UI、REQUEST_HUMAN_INPUT 弹出原生确认框。
- **资源**：
  - 官方博客教程：https://www.copilotkit.ai/blog/ag-ui-goes-mobile-the-kotlin-sdk-unlocks-full-agent-connectivity-across-android-ios-and-jvm
  - AG-UI 文档：https://docs.ag-ui.com （搜索 Kotlin SDK 示例）

如果你想复用现有 React UI 代码：

- **次选方案：WebView 嵌入**（简单但有性能折衷）
  - 在 Android 用 WebView，iOS 用 WKWebView。
  - 打包你的 React 应用（用 Next.js 或 Vite 构建静态文件）。
  - 在原生 App 中加载本地 HTML/JS 文件，或远程 URL。
  - 通过 JS Bridge（addJavascriptInterface / WKScriptMessageHandler）传递原生上下文给 React。
  - 缺点：有 JS 桥接开销，不如纯原生流畅；需处理权限、安全。

不推荐直接用 React Native：目前 @copilotkit/react-ui 没有官方 React Native 支持（GitHub Issue #1892 仍在讨论），虽有人尝试 polyfill crypto/uuid，但 UI 组件需重写。

2. Linux 桌面应用

- **推荐方案：Electron 包装 Web 应用**（最简单、跨平台）
  - 用 Electron 构建桌面 App，内部嵌入你的 React + @copilotkit/react-ui 应用。
  - 支持 Linux、Windows、macOS 一份代码。
  - 步骤：
    1. 用 Create React App / Next.js 构建你的 CopilotKit 前端。
    2. 用 Electron 包装（electron-forge 或 electron-builder）。
    3. 发布 .AppImage / .deb 等 Linux 格式。
  - 示例：社区已有类似 Microsoft Copilot 的 Electron 包装项目，可参考改造。
  - 优势：直接复用全部 React UI 代码，无需改动。
- **纯原生方案**（如果追求极致性能）：
  - 用 Tauri（Rust + WebView）或 Neutralinojs 替代 Electron（更轻量）。
  - 或者用 AG-UI 的其他语言 SDK（如 Go、Rust）构建原生 UI（但需手动实现聊天组件）。
- **WebView 替代**：在 GTK/Qt 原生 App 中嵌入 WebView（WebKitGTK），加载 React 构建产物。

总结对比表

平台	推荐方式	是否复用 React UI 代码	性能/原生度	难度	资源链接
Android/iOS	AG-UI Kotlin SDK（原生）	部分（逻辑复用，UI 重写）	高（无桥接）	中等	Kotlin SDK 博客
Android/iOS	WebView 嵌入	完全复用	中等	低	Android WebView / iOS WKWebView 文档
Linux	Electron 包装	完全复用	中等	低	Electron.js 官网
Linux	Tauri / 原生 WebView	完全复用	高	中等	Tauri.dev

建议：如果你追求最佳原生体验和未来扩展性，优先迁移到 **AG-UI Kotlin SDK**（移动端），后端保持不变，前端用 Compose/SwiftUI 重建 UI（代码量不大，因为 CopilotKit UI 简单）。桌面 Linux 用 Electron 最快上线。

如果你有具体场景（如现有 App 是 Kotlin/Swift，还是想完全复用 React 代码），告诉我细节，我可以提供更精确的代码模板或迁移指南！🚀

CopilotKit 是否有支持 React Native 的库？（2025 年 12 月最新情况） 没有官方支持的 React Native 专用库。 CopilotKit 的核心包（如 @copilotkit/react-ui、@copilotkit/react-core）是专为 Web React（包括 Next.js）设计的 UI 组件和 Hook（如 CopilotChat、CopilotSidebar、useAgent），它们依赖浏览器环境（如 DOM、CSS），直接在 React Native 中运行会遇到兼容性问题（例如 crypto、uuid 等模块需 polyfill，UI 组件需重写）。 当前社区与官方状态

GitHub 上有一个 Feature Request Issue #1892（2025 年 5 月提出）：用户请求 React Native 支持，并表示愿意贡献。开发者提到通过 Expo + React Native Web + Metro bundler 可以让核心功能运行（只需 polyfill crypto 和 uuid），但 UI 组件仍需迁移为 React Native 原生版本。目前该 Issue 仍开放，未合并任何官方 RN 支持。 官方文档、npm 包描述、博客和 GitHub Releases 中均未提及 React Native 支持，所有示例均为 Web React。 CopilotKit 强调与 AG-UI 协议 的集成，支持多框架前端（如 React、Angular），但移动端目前通过社区 Kotlin Multiplatform SDK（Android/iOS 原生）实现，而不是 React Native。

React Native 下的替代方案 如果你想在 React Native App 中集成 CopilotKit 的 Agentic 功能（实时聊天、状态同步、Human-in-the-Loop）：

推荐：使用 AG-UI 协议手动实现（不依赖 React UI 包） 后端保持不变（暴露 AG-UI 端点）。 在 React Native 中用 WebSocket 或 HTTP SSE 连接 AG-UI 事件流。 手动处理事件（TEXT_MESSAGE_CONTENT 显示消息、STATE_DELTA 更新状态、REQUEST_HUMAN_INPUT 弹出原生对话框）。 用 React Native 原生组件（如 Gifted Chat、React Native Paper）构建聊天 UI。 优势：完整 Agentic 特性，性能好。 参考：AG-UI 文档（https://docs.ag-ui.com）有事件规范，可参考社区 Kotlin SDK 的逻辑移植。

实验性：React Native Web + Expo 如 Issue 中提到的，有人成功在 Expo 中运行核心逻辑（polyfill 后）。 但 @copilotkit/react-ui 的预置组件（如 Popup/Sidebar）在 RN 中渲染效果差，需要自定义样式或替换为 RN 组件。 适合快速原型，不推荐生产。

WebView 嵌入（简单但非原生） 在 React Native App 中用 加载你的 Web React + CopilotKit 页面。 通过 postMessage 桥接上下文。 缺点：性能和体验不如原生。

总结 目前（2025 年 12 月），CopilotKit 没有官方 React Native 库，社区也在等待或自行尝试。如果你对 React Native 支持感兴趣，可以关注/参与 GitHub Issue #1892，或者考虑切换到 AG-UI 的 Kotlin SDK（纯原生移动支持，已在 2025 年 11 月发布）。 如果你有具体的 React Native 场景（比如想实现聊天 + 状态同步），我可以帮你写一个基于 AG-UI 的手动集成示例代码！🚀