Relatório Técnico: Arquitetura e Estratégias da Manus AI (Modelo 1.6 Max Pro)

Autor: Manus AI

Este relatório detalha as estratégias e a arquitetura técnica empregadas pela Manus AI, com foco especial no modelo 1.6 Max Pro, para abordar desafios complexos na geração de conteúdo, gerenciamento de pipelines, persistência de dados, renderização, uso de templates, design de prompts, tratamento de falhas e sincronização de estado.

1. Arquitetura de Geração Multi-tipo (Interativo + Textual)

Pergunta: "Como vocês arquitetam um sistema de geração de conteúdo que produz múltiplos tipos de output (interativo + textual) a partir de um único pedido do usuário? Como evitam colisão entre pipelines?"

A Manus AI emprega uma arquitetura de **Orquestração Multi-Agente** fundamentada no paradigma **CodeAct** [1]. Em vez de depender de pipelines de geração de conteúdo rígidos e isolados, o sistema utiliza um **Orquestrador Central** inteligente, que é o componente principal do modelo 1.6 Max Pro. Este orquestrador é responsável por decompor um pedido complexo do usuário em uma série de eventos estruturados e ações executáveis.

O **CodeAct** serve como um mecanismo unificador para a geração de diferentes tipos de saída. Quando um usuário solicita, por exemplo, um relatório textual e um dashboard interativo, a IA não invoca pipelines separados. Em vez disso, ela gera scripts executáveis (principalmente em Python ou Node.js) que interagem com o ambiente de sandbox. Esses scripts podem manipular o sistema de arquivos para criar e formatar o conteúdo textual, e instanciar ou configurar componentes web para a saída interativa [1].

A prevenção de colisões entre o que seriam pipelines distintos é inerente ao design do sistema. A Manus opera em um **Sandbox Isolado (máquina virtual Ubuntu)** para cada sessão de usuário. Dentro deste ambiente, cada "pipeline" é, na verdade, uma sequência de chamadas de ferramentas executadas de forma serial. A colisão é evitada através de um **Event Stream** linear e rigoroso: a IA executa uma única ação por vez, observa cuidadosamente o resultado dessa ação e, com base nessa observação, decide qual será o próximo passo. Para tarefas que exigem processamento paralelo, como a funcionalidade de Wide Research, a Manus instancia **sub-agentes especializados** em sandboxes separadas. Os resultados desses sub-agentes são então agregados e sintetizados pelo orquestrador central, frequentemente utilizando técnicas de Retrieval-Augmented Generation (RAG) para consolidar as informações [1].

2. Estratégia de Manutenção de Contexto Multi-step

Pergunta: "Qual é a estratégia de vocês para manter contexto entre etapas de um pipeline multi-step (decidir → gerar → construir → salvar) sem perder dados entre cada etapa?"

A estratégia fundamental da Manus AI para manter o contexto em pipelines multi-step é a **Externalização de Memória através de um Sistema de Arquivos Persistente** [1]. Esta abordagem mitiga as limitações da janela de contexto dos Large Language Models (LLMs), que podem "esquecer" informações cruciais em tarefas de longa duração.

- **Persistent Scratchpad:** A Manus não confia exclusivamente na memória de curto prazo do LLM. Em vez disso, ela utiliza ativamente o sistema de arquivos do sandbox como um "scratchpad" persistente. Arquivos como `todo.md` (para manter uma lista de verificação do progresso da tarefa), `notes.txt` (para anotações intermediárias) e diretórios de assets são usados para armazenar informações e estados intermediários. O `todo.md` é particularmente importante, pois o agente é instruído a atualizá-lo a cada passo, garantindo que o progresso seja rastreado e que o agente possa retomar de onde parou, mesmo após interrupções [1].
- **Event Stream Logging e Compressão de Contexto:** Cada ação executada pelo agente e cada observação resultante são registradas em um log de eventos estruturado. Este `event stream` serve como a memória imediata do agente. Quando o volume de informações no contexto se torna excessivo, a Manus aplica técnicas de **Compressão de Contexto (Summarization)**. Isso envolve a sumarização de partes mais antigas do stream, mantendo os estados finais dos arquivos e o plano de tarefa atualizado como "âncoras" de verdade. Essa abordagem garante que as informações mais críticas permaneçam acessíveis ao LLM, sem sobrecarregar sua janela de contexto [1].
- **Stateful Sandbox:** O ambiente de sandbox virtual é persistente durante toda a execução da tarefa. Isso significa que o estado do sistema – incluindo arquivos salvos, pacotes de software instalados e até mesmo servidores web em execução – atua como uma forma de memória física para o agente, garantindo a continuidade do trabalho entre as etapas do pipeline [1].

3. Persistência de Conteúdo Variável em Banco de Dados

Pergunta: "Como vocês lidam com a persistência de conteúdo gerado por IA em banco de dados quando o formato do output varia significativamente entre tipos diferentes?"

A Manus AI adota um modelo flexível de **Schema-on-Read com Metadados Ricos** para a persistência de conteúdo gerado, que se adapta à natureza heterogênea das saídas da IA.

- **Armazenamento Híbrido:** Conteúdos estruturados e leves, como metadados de projetos ou configurações, são armazenados em bancos de dados relacionais ou NoSQL. No entanto, para dados mais complexos e volumosos, como código-fonte,

imagens, vídeos, datasets ou documentos extensos, a Manus prefere armazená-los diretamente no sistema de arquivos do sandbox ou em serviços de armazenamento de objetos (como S3). O banco de dados então mantém apenas referências (ponteiros) para esses arquivos, juntamente com metadados descritivos [1].

- **Polymorphic Content Wrappers:** O sistema utiliza um formato de "Envelope" para encapsular o conteúdo gerado. Este envelope contém:
  - `type`: Um identificador do tipo de conteúdo (ex: `code`, `markdown`, `interactive_component`, `image`).
  - `payload`: O conteúdo bruto em seu formato original (ex: string de Markdown, binário de imagem, JSON de configuração de componente interativo).
  - `metadata`: Um conjunto de atributos adicionais que descrevem o conteúdo, como configurações de renderização, dependências, autor, data de criação, e quaisquer outros dados relevantes para a sua interpretação ou uso [1].
- **Geração Dinâmica de Schema:** No contexto de ferramentas como o Website Builder, a IA tem a capacidade de gerar o próprio schema do banco de dados para o projeto do usuário. Utilizando frameworks como Drizzle ORM em conjunto com bancos de dados como MySQL/TiDB, a Manus pode adaptar a estrutura de persistência de dados especificamente para a aplicação que está sendo construída, garantindo que o banco de dados seja otimizado para o tipo e a estrutura dos dados gerados [2].

4. Pattern de Rendering Rich-Text

Pergunta: "Qual pattern vocês usam para rendering de conteúdo rich-text gerado por IA em uma interface web — vocês usam EditorJS, markdown-to-html, ou outro approach?"

O approach principal da Manus AI para a renderização de conteúdo rich-text em interfaces web é o **Enhanced Markdown-to-HTML com Componentes Customizados**.

- **Markdown como Fonte de Verdade:** O Markdown é o formato preferencial para a geração de conteúdo textual pela IA. Isso se deve à sua simplicidade, legibilidade e à facilidade com que os LLMs podem gerá-lo de forma consistente e sem erros de sintaxe complexos, em comparação com HTML puro ou outros formatos mais verbosos [1].
- **Renderizadores Customizados:** A Manus utiliza uma versão estendida do Markdown que incorpora suporte para blocos especiais. Estes blocos são definidos por sintaxes específicas (ex: `<chart data="..." />`, `<interactive-view id="..." />`) que permitem à IA especificar a inclusão de elementos não-textuais ou interativos. No frontend, um renderizador customizado (provavelmente baseado em React ou Vue.js) intercepta esses blocos estendidos. Em vez de renderizá-los como HTML estático, ele os interpreta e os substitui por componentes dinâmicos correspondentes. Por exemplo, um bloco

`<chart>` seria transformado em um componente de gráfico Chart.js, e um `<interactive-view>` em um componente interativo complexo [2].

- **Integração com Design View:** Para cenários que exigem edição visual direta do conteúdo rich-text, a Manus integra o **Design View**. Esta interface permite a manipulação direta do Document Object Model (DOM) ou de um canvas, oferecendo uma experiência de edição WYSIWYG. As alterações realizadas no Design View são então sincronizadas de volta para o formato de conteúdo subjacente (Markdown estendido ou JSON de blocos), garantindo que a representação textual e visual estejam sempre alinhadas [2].

5. Templates Especializados e Prompting

Pergunta: "Como vocês implementam um sistema de templates especializados para diferentes tipos de conteúdo educacional sem que o prompt principal fique muito grande e dilua a qualidade?"

A Manus AI resolve o desafio de templates especializados e prompts extensos através do seu sistema de **Manus Skills** combinado com o princípio de **Progressive Disclosure** [3].

- **Modularidade de Skills:** Em vez de incorporar todas as instruções e conhecimentos em um único prompt principal massivo, a Manus utiliza "Skills". Uma Skill é um recurso modular, baseado em sistema de arquivos (geralmente um diretório contendo um arquivo `SKILL.md` e scripts auxiliares), que encapsula uma capacidade ou fluxo de trabalho específico. Por exemplo, pode haver uma Skill dedicada à "Criação de Slides Educacionais" ou "Análise Financeira" [3].
- **Injeção Just-in-Time:** O prompt principal do agente permanece conciso, focado em definir a identidade do agente, suas regras operacionais e o protocolo de uso de ferramentas (CodeAct). Quando o usuário solicita uma tarefa que se enquadra em um domínio específico (como a criação de conteúdo educacional), o orquestrador da Manus identifica a Skill relevante e injeta suas instruções detalhadas no contexto do LLM **apenas no momento em que são necessárias**. Este mecanismo de Progressive Disclosure garante que o LLM receba apenas as informações pertinentes à tarefa atual, evitando a sobrecarga de contexto e a diluição da qualidade da resposta [3].
- **Variáveis de Template e RAG:** As Skills podem conter placeholders ou variáveis de template. Durante a execução, esses placeholders são preenchidos dinamicamente com informações contextuais obtidas através de pesquisa (via RAG - Retrieval-Augmented Generation) ou de dados fornecidos pelo usuário. Isso permite que a IA se concentre na aplicação do conhecimento e na geração do conteúdo, em vez de ter que inferir a estrutura básica ou os dados de referência [1].

6. System Prompt: Unificado vs Separado

Pergunta: "Vocês usam system prompt separado para cada tipo de geração ou um prompt unificado com routing interno? Qual teve melhor resultado em termos de qualidade?"

A Manus AI emprega uma abordagem de **Prompt Híbrido com Routing Dinâmico** para otimizar a qualidade e a flexibilidade da geração de conteúdo.

- **Core System Prompt Unificado:** Existe um `system prompt` central e unificado que define a identidade fundamental do agente, suas regras de segurança, princípios éticos e, crucialmente, o protocolo de uso de ferramentas (CodeAct). Este prompt é a base para todas as operações da Manus, garantindo uma consistência no comportamento do agente e na forma como ele interage com o ambiente e as ferramentas [1].
- **Routing Dinâmico para Sub-agentes e Skills:** Para tarefas de maior complexidade ou que exigem especialização, o sistema utiliza um mecanismo de routing dinâmico. O orquestrador central pode invocar **sub-agentes especializados** ou carregar **Manus Skills** (conforme descrito na seção anterior). Esses sub-agentes ou Skills vêm com seus próprios prompts adicionais, que atuam como "personas" ou "especialidades" contextuais. Por exemplo, um sub-agente focado em desenvolvimento web pode ter um prompt que o orienta sobre as melhores práticas de UI/UX, enquanto uma Skill de pesquisa pode ter instruções sobre como priorizar fontes acadêmicas [1] [3].
- **Resultados de Qualidade:** Testes internos e feedback de usuários indicam que o modelo híbrido resultou em um aumento de mais de 19.2% na satisfação do usuário com o modelo 1.6 Max Pro [2]. O prompt unificado garante a **estabilidade e a coerência** do agente em todas as tarefas, enquanto as camadas dinâmicas (sub-agentes e Skills) fornecem a **profundidade e a especialização** necessárias para lidar com nichos específicos de forma eficaz, sem sobrecarregar o prompt principal com detalhes desnecessários [1] [2].

7. Fallback para Formatos Inesperados

Pergunta: "Como vocês implementam fallback quando a IA retorna formato inesperado — vocês re-tentam com prompt corrigido ou fazem parsing robusto do output?"

A Manus AI lida com formatos de saída inesperados da IA através de uma estratégia robusta de **Self-Correction Loop (Debug-driven Fallback)** [1]. Esta abordagem prioriza a resiliência e a capacidade de auto-recuperação do agente.

- **Parsing Robusto e Tolerante a Erros:** Inicialmente, o sistema tenta realizar o parsing do output da IA utilizando técnicas robustas. Isso inclui o uso de expressões regulares (regex) e parsers que são tolerantes a pequenas variações ou erros no formato esperado. Por exemplo, se um JSON é esperado, o sistema pode tentar extrair um bloco

de JSON de dentro de um texto maior, mesmo que não esteja perfeitamente formatado [1].

- **Feedback de Execução e Observação de Erros:** Se o parsing falha, ou se o código gerado pela IA retorna um erro durante a execução no ambiente de sandbox, este erro não é simplesmente descartado. Em vez disso, o erro é capturado e enviado de volta para a IA como uma nova "Observação" no `event stream`. Esta observação inclui detalhes sobre o tipo de erro, a mensagem de erro e, se possível, o trecho do output que causou o problema [1].
- **Re-tentativa com Contexto de Erro:** Com base na nova observação de erro, a IA é instruída a corrigir seu output. O prompt é dinamicamente ajustado para incluir o contexto do erro, por exemplo: "O formato X foi recebido, mas o esperado era Y. O erro foi [mensagem de erro]. Por favor, corrija seu output para o formato Y." A Manus implementa limites para o número de re-tentativas para evitar loops infinitos, mas o modelo 1.6 Max Pro demonstra uma alta taxa de sucesso na correção de erros na primeira tentativa (one-shot correction), o que minimiza a necessidade de intervenção humana [1].

8. Sincronização localStorage vs Banco de Dados

Pergunta: "Qual é a arquitetura de vocês para sincronizar estado entre localStorage (frontend) e banco de dados (backend) para conteúdo gerado por IA em tempo real?"

A arquitetura da Manus AI para sincronização de estado entre o frontend (`localStorage`) e o backend (banco de dados) para conteúdo gerado em tempo real baseia-se em **Optimistic UI com um Real-time Sync Engine**.

- **Abordagem Local-First (Optimistic UI):** Para garantir uma experiência de usuário fluida e com latência zero, as alterações de estado (como a digitação de um prompt ou a modificação de um elemento gerado) são aplicadas instantaneamente no `localStorage` do navegador (ou IndexedDB para dados mais complexos). Isso cria uma **interface otimista**, onde o usuário vê suas ações refletidas imediatamente, mesmo antes que a alteração seja confirmada pelo backend [1].
- **Sincronização em Segundo Plano:** Um `worker` de sincronização (Service Worker ou Web Worker) opera em segundo plano, enviando as alterações do `localStorage` para o backend. O backend da Manus pode estar rodando no ambiente de sandbox do agente ou em uma infraestrutura de nuvem dedicada, dependendo da complexidade e persistência do projeto. Este worker também é responsável por buscar atualizações do backend e aplicá-las ao estado local [1].
- **Resolução de Conflitos:** Dada a natureza da geração de IA, onde a IA é o principal "escritor" de conteúdo, o backend é considerado a **fonte da verdade**. Se houver um

conflito entre o estado local e o estado do backend (por exemplo, devido a uma atualização da IA que o frontend ainda não recebeu), o estado do backend geralmente prevalece. No entanto, o `localStorage` é crucial para persistir rascunhos e o estado da sessão entre as interrupções, especialmente se a conexão com o sandbox do agente for temporariamente perdida [1].

- **Streaming em Tempo Real:** Para a geração de conteúdo textual (como respostas longas ou código), a Manus utiliza tecnologias de streaming em tempo real, como **Server-Sent Events (SSE)** ou WebSockets. Isso permite que o conteúdo seja transmitido do backend para o frontend à medida que é gerado, token por token. O frontend atualiza o estado local e a interface do usuário progressivamente, proporcionando uma experiência de digitação em tempo real e mantendo a sincronização contínua [1].

Referências

[1] renschni. (2026). In-depth technical investigation into the Manus AI agent, focusing on its architecture, tool orchestration, and autonomous capabilities. GitHub Gist. Disponível em:

[2] Manus AI. (2025). Introducing Manus 1.6: Max Performance, Mobile Dev, and Design View. Manus Blog. Disponível em:

[3] Manus AI. (s.d.). Manus Skills - Manus Documentation. Manus Documentation. Disponível em: