



超越界面：前端工程师如何塑造 AI 原生应用的未来

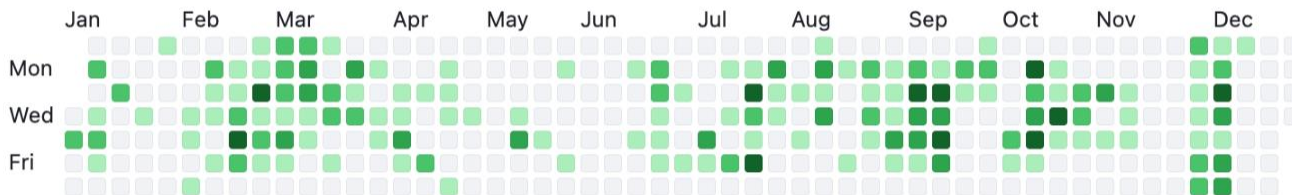
Beyond UI: How Frontend Engineers Can Shape the Future of AI-Native Applications

About Me



Frontend Software Engineer @ Workflow & RAG Squad

Contributors 1,078



WTW0313



twwu@dify.ai



Th3_W0R1d

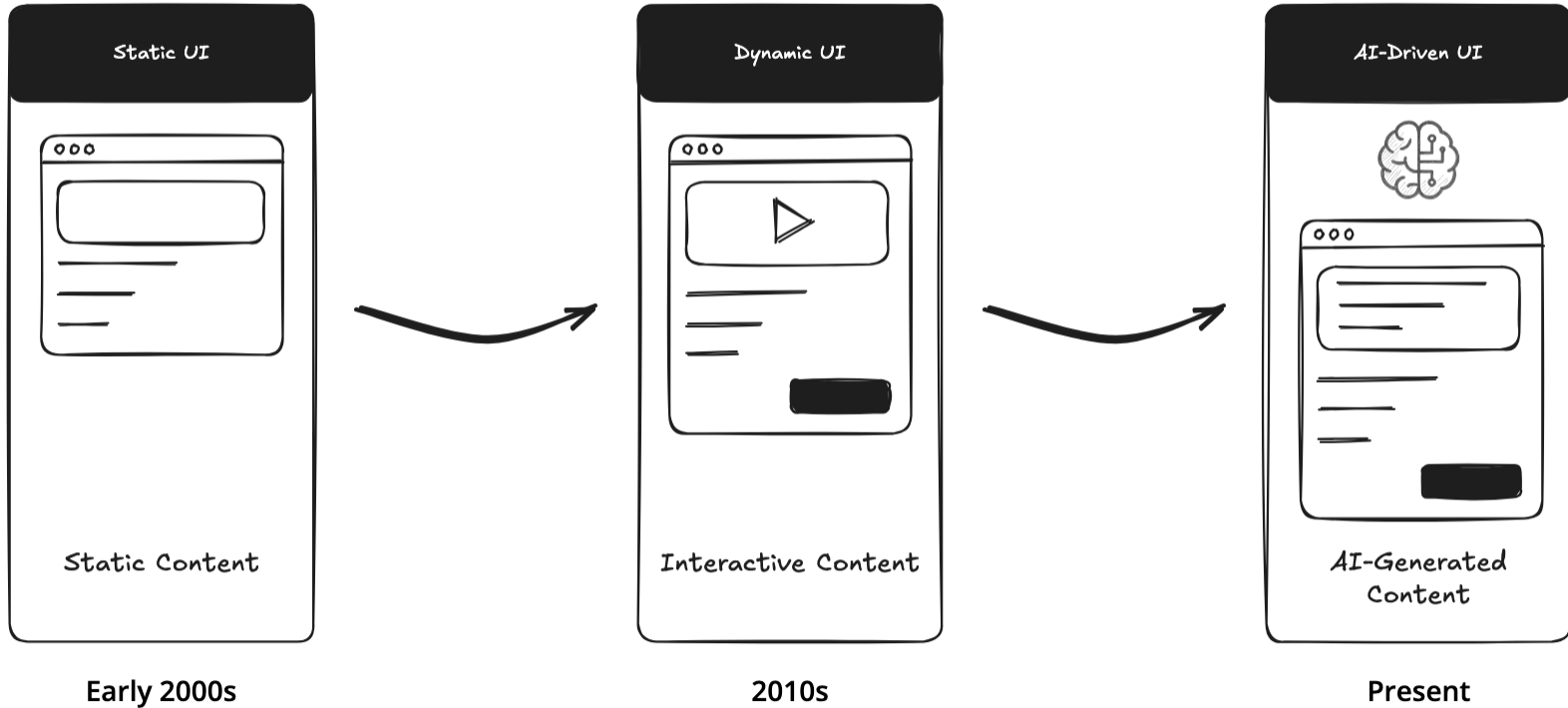
Outline

30 min · 4 parts | 5 min · Q&A

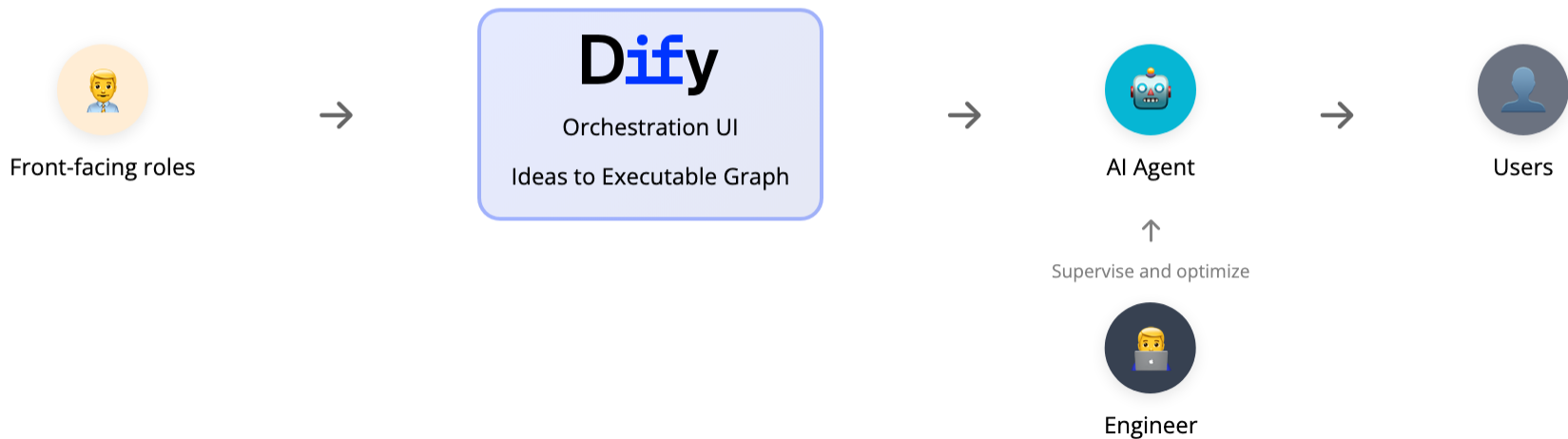
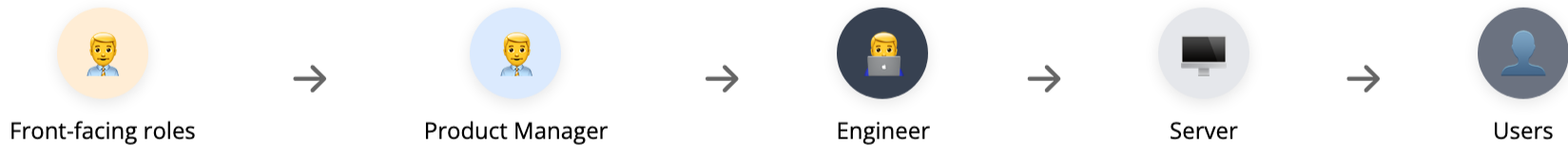
- **Part 1 — The Evolution of Frontend Development**
From **deterministic** to **probabilistic**
- **Part 2 — Dify**
Why **workflow** is a good choice?
- **Part 3 — Frontend Design & Architecture Patterns**
All for **extensibility**
- **Part 4 — Live Demo**
Use **Workflow & Knowledge Pipeline** to build an AI application in a few minutes

The Evolution of Frontend Development

From deterministic to probabilistic



From Expert to Everyone



Agentic Workflow

Drag and drop, orchestrate the nodes, and turn your AI vision into reality.

The screenshot displays the Dify AI Studio interface, which is used for building and testing AI workflows. The top navigation bar includes the Dify logo, a 'Solar Studio' dropdown menu, a 'TEAM' button, and links to 'EXPLORE', 'STUDIO / CHATBOT PLAYGROUND', 'KNOWLEDGE', and 'TOOLS'. On the right side of the top bar, there are links for 'PLUGINS' and a user profile icon.

The main workspace shows a workflow canvas with several nodes connected in a sequence. The nodes are:

- DOC EXTRACTOR**: A node for extracting content from documents. It has a 'VARIABLE' input field and a 'sys.files File' input.
- LLM**: A node for invoking large language models. It is configured with 'GPT-4o' as the model and has a description: 'Invoking large language models to answer questions or process natural language'.
- CODE**: A node for executing code. It has a 'Fail Branch' output.

The workflow is currently in an 'Auto-Saved 21:02:35 · Unpublished' state. A 'Test run' button is visible in the top right corner of the workflow area.

On the right side, a 'Test Run#5' panel is open, showing the results of a test run. The panel has tabs for 'INPUT', 'RESULT', 'DETAIL', and 'TRACING'. The 'TRACING' tab is selected, showing a list of steps in the workflow:

- START**: 0.02 s, status: success (green checkmark).
- DOC EXTRACTOR**: 0.27 s, status: success (green checkmark).
- LLM**: 258 tokens · 0.12 s, status: success (green checkmark).
- CODE**: 1.53 s, status: error (yellow warning triangle).

The 'CODE' node's error message is displayed in an orange box: 'Node exception, will automatically execute the fail branch. The node output will return an error type and error message and pass them to downstream.'

The 'CODE' node's input and output fields are also visible. The input field contains a Jinja template: `01 { 02 "query": "What model are you?" 03 }`. The output field is currently empty.

Knowledge Pipeline

Visually customize your data processing flow for better accuracy and relevance.

The screenshot displays the Dify Knowledge Pipeline interface. At the top, the navigation bar includes the Dify logo, a project name 'Solar Studio' with a 'TEAM' tag, and tabs for 'EXPLORE', 'STUDIO', 'KNOWLEDGE', and 'TOOLS'. The 'KNOWLEDGE' tab is active, showing a specific knowledge base named 'CANON EOS CAMERA S...'. Below the navigation bar, the main workspace shows a visual pipeline flow:

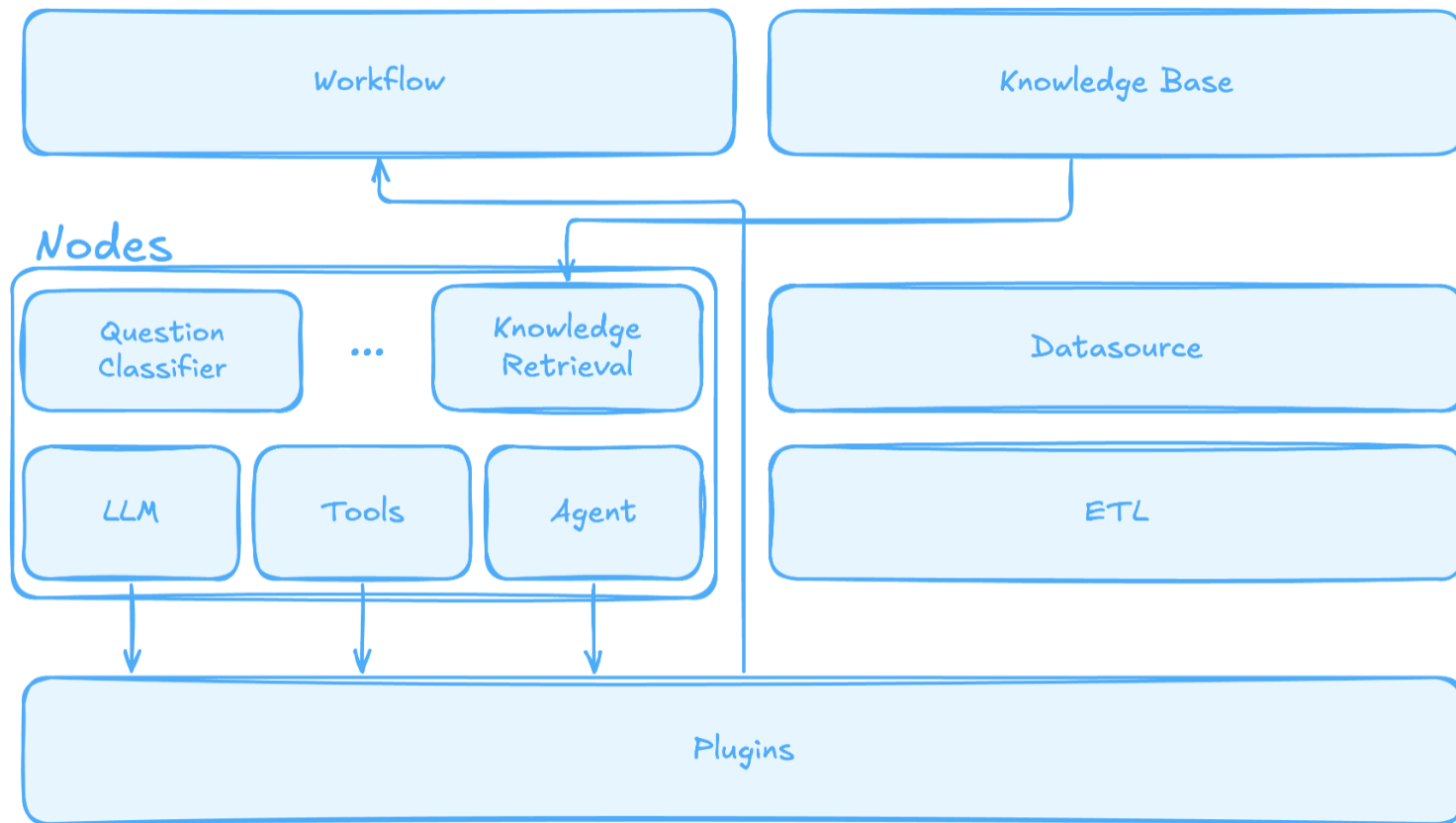
- DATA SOURCE:** A list on the left includes 'FILE UPLOAD' (Upload local files), 'NOTION' (Sync Notion pages), and 'JINA READER'.
- DOC EXTRACTOR:** A central node with an 'INPUT VARIABLE' section showing 'File Upload' and '(x) documents' File.
- GENERAL CHUNKER:** A node with an 'INPUT VARIABLE' section showing 'Doc Extractor' and '(x) content' File.
- KNOWLEDGE BASE:** A node on the right with an 'INDEX METHOD' section showing 'Doc Extractor' and '(x) content' String.

On the right side, a panel titled 'Knowledge Base' provides configuration options:

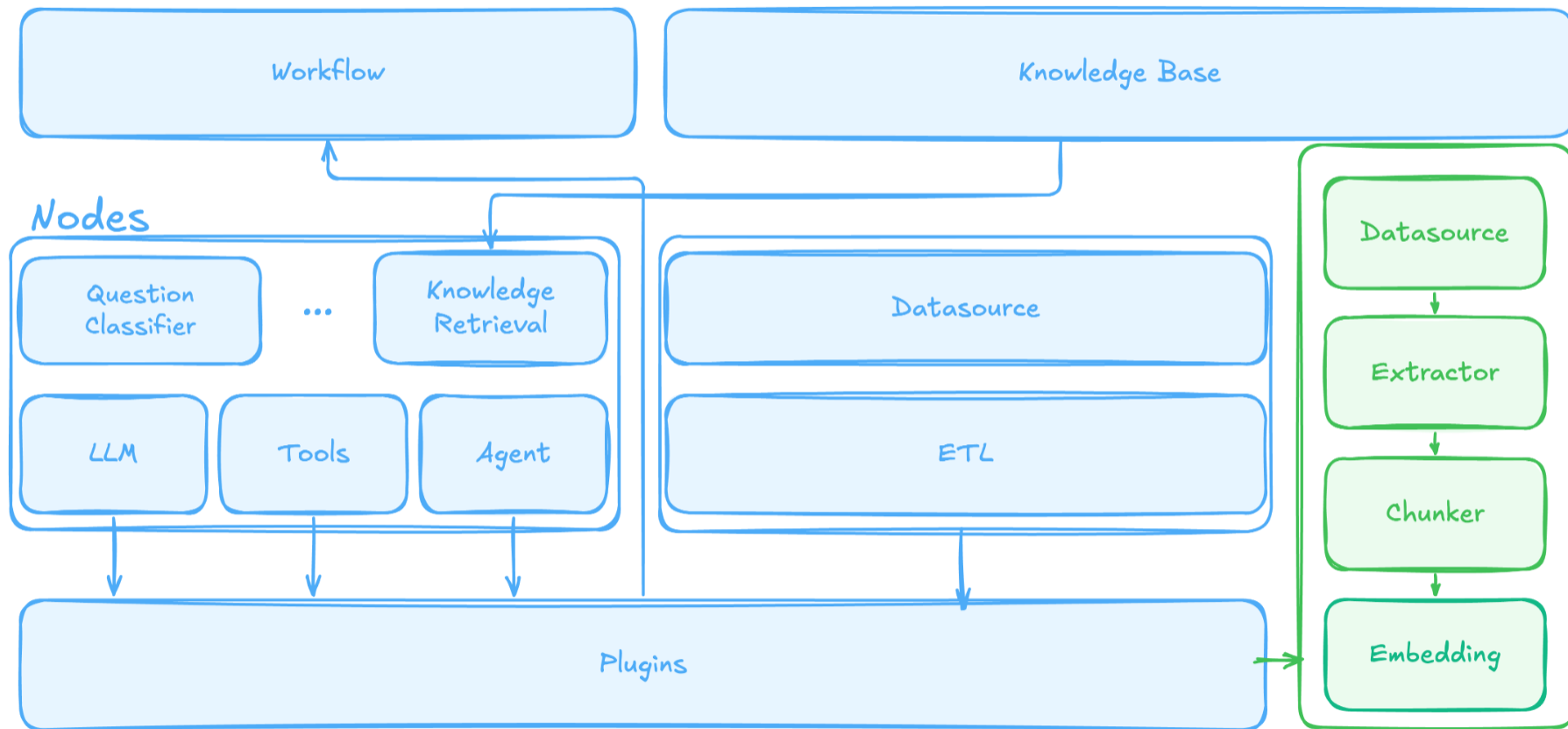
- CHUNK STRUCTURE:** A dropdown menu set to 'General' (General text chunking mode, the chunks retrieved and recalled are the same).
- INPUT VARIABLE:** A dropdown menu set to 'Doc Extractor' and '(x) content' String.
- INDEX METHOD:** A dropdown menu set to 'High Quality' (Call default system embedding interface for processing to provide higher accuracy when users query).

At the top right of the workspace, there are buttons for 'Input field', 'Test run', and 'Publish'.

Original Workflow & RAG System



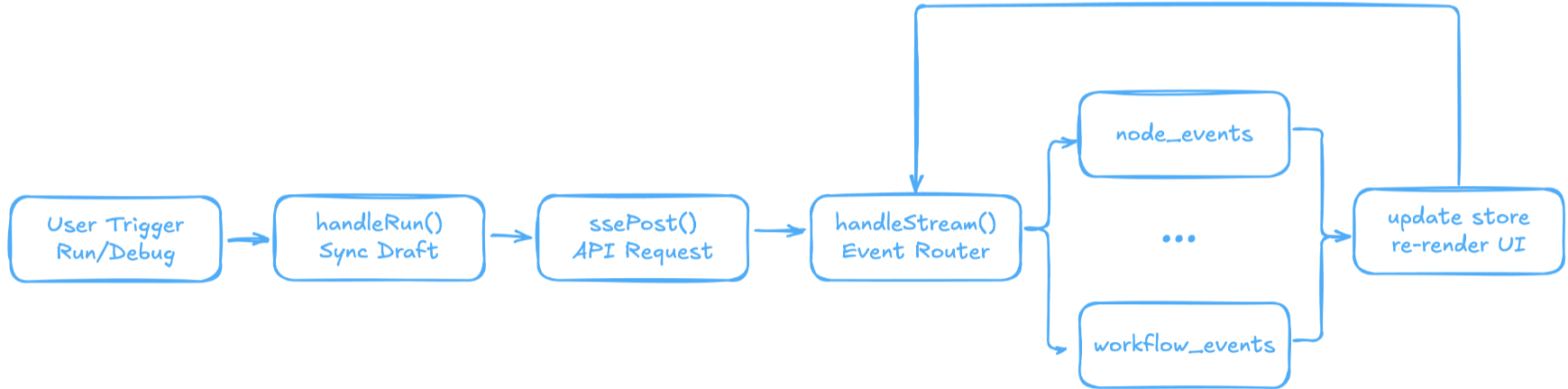
New Workflow & RAG System



Visual Workflow Builder

Features	Implementation
Canvas	ReactFlow with custom node/edge types
State Management	Zustand with sliced stores
Real-time	SSE streaming for execution events
Debugging	Built-in debug panel for tracing execution
History	Version control and draft management

Execution Lifecycle



One Workflow Builder
Multiple Applications

Store Slice Injection

```
export type SliceFromInjection
  = Partial<WorkflowAppSliceShape>
  & Partial<RagPipelineSliceShape>

export const createWorkflowStore = (params: CreateWorkflowStoreParams) => {
  const { injectWorkflowStoreSliceFn } = params || {}

  return createStore<Shape>((...args) => ({
    ...createChatVariableSlice(...args),
    ...createEnvVariableSlice(...args),
    ...createNodeSlice(...args),
    ...createWorkflowSlice(...args),
    // ... more base slices
    ...(injectWorkflowStoreSliceFn?.(...args) || {} as SliceFromInjection),
  })))
}
```

Core store stays **stable**

Apps inject only **what they own**

Store Slice Injection

```
export type SliceFromInjection
  = Partial<WorkflowAppSliceShape>
  & Partial<RagPipelineSliceShape>

export const createWorkflowStore = (params: CreateWorkflowStoreParams) => {
  const { injectWorkflowStoreSliceFn } = params || {}

  return createStore<Shape>((...args) => ({
    ...createChatVariableSlice(...args),
    ...createEnvVariableSlice(...args),
    ...createNodeSlice(...args),
    ...createWorkflowSlice(...args),
    // ... more base slices
    ...(injectWorkflowStoreSliceFn?.(...args) || {} as SliceFromInjection),
  })))
}
```

Core store stays **stable**

Apps inject only **what they own**

Workflow

```
export type WorkflowSliceShape = {
  appId: string
  appName: string
  notInitialWorkflow: boolean
  showOnboarding: boolean
  nodesDefaultConfigs: Record<string, any>
  // ... workflow-app specific state
}

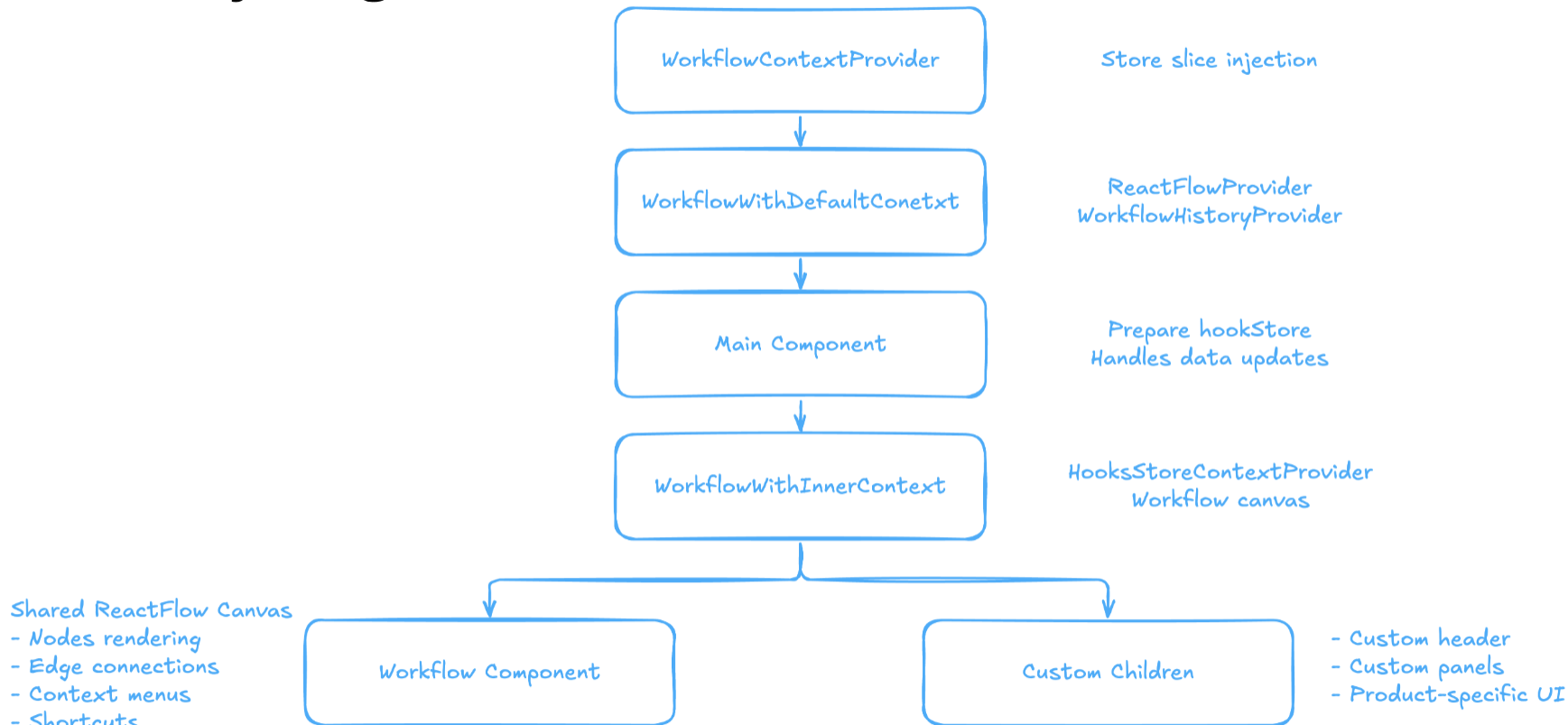
const WorkflowAppWrapper = () => {
  return (
    <WorkflowContextProvider
      injectWorkflowStoreSliceFn={createWorkflowSlice}
    >
      <WorkflowAppWithAdditionalContext />
    </WorkflowContextProvider>
  )
}
```

Knowledge Pipeline

```
export type RagPipelineSliceShape = {
  pipelineId: string
  knowledgeName: string
  knowledgeIcon?: IconInfo
  showInputFieldPanel: boolean
  ragPipelineVariables: RAGPipelineVariables
  dataSourceList: ToolWithProvider[]
  // ... RAG-specific state
}

const RagPipelineWrapper = () => {
  return (
    <WorkflowContextProvider
      injectWorkflowStoreSliceFn={createRagPipelineSlices}
    >
      <RagPipeline />
    </WorkflowContextProvider>
  )
}
```

Hierarchy Diagram



Basic Types

- `string`
- `number`
- `boolean`
- `file`
- `array`
 - `array[string]`
 - `array[number]`
 - `array[boolean]`
 - `array[file]`
 - `array[object]`
- `object`

Special Types



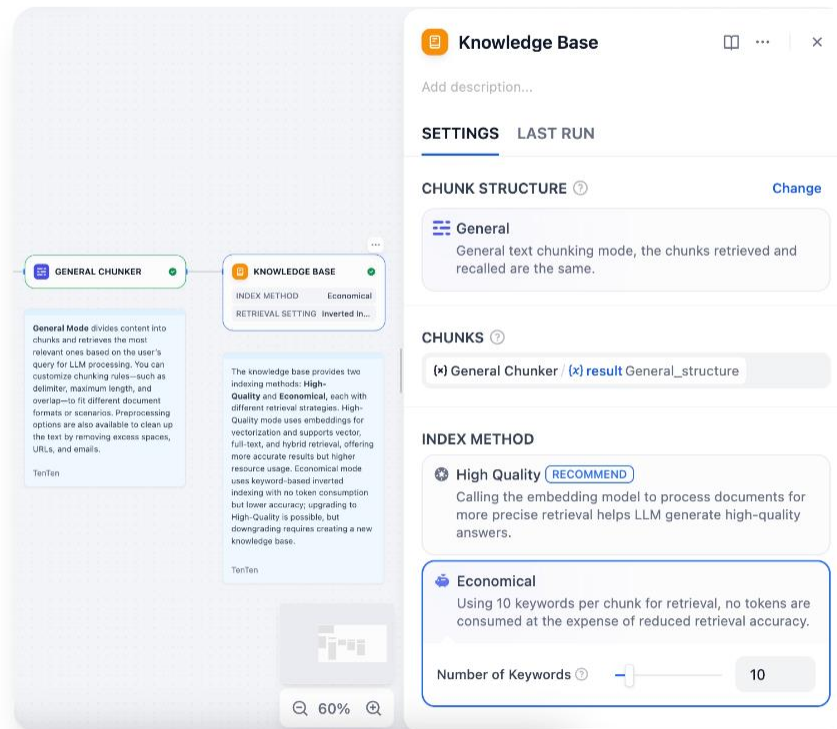
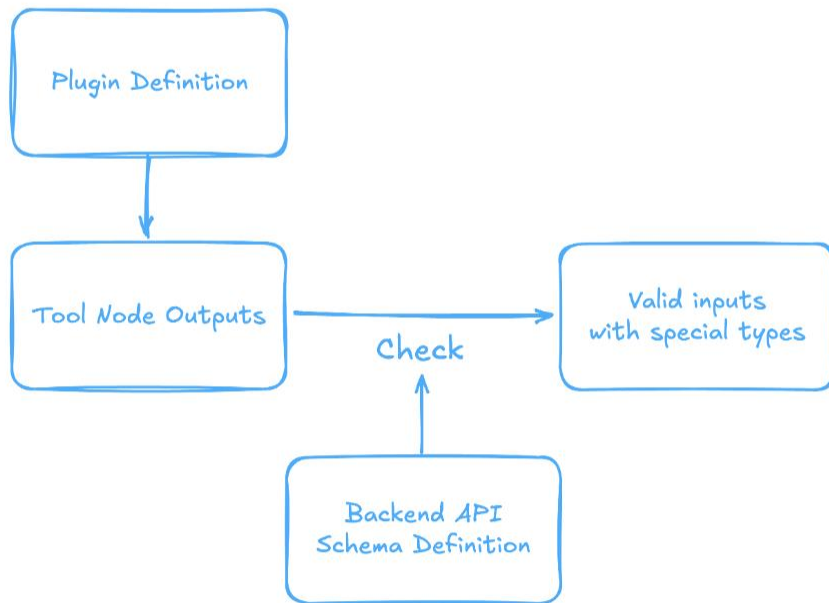
General Chunker

`array[string]`

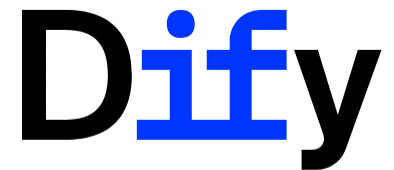
`general_structure`

`array[string]`

Solution



Live Demo



Thank you

WEBSITE

dify.ai

GITHUB

github.com/langgenius/dify

JOIN US

joinus@dify.ai