

# PRD: A Python TUI Primitive for Coding Agents

Status: Draft v0.1 Author: Arvind Working codename: `scrollback` (final name TBD) Last updated: May 1, 2026

---

## 1. Problem

Python is the dominant ecosystem for AI agents — LangGraph, smol-agents, OpenHands, Aider, the entire HuggingFace stack, every research codebase. But when a Python developer wants to put a polished terminal UI on a coding agent, they hit a fork in the road with no good middle path:

- **Textual** is a full TUI framework optimized for multi-pane applications. It owns the alt-buffer, breaks native terminal selection, and forces a CSS/widget mental model that's overkill for a chat-shaped agent loop.
- **prompt\_toolkit + Rich** in REPL mode preserves scrollback and copy-paste, but offers no agent-specific primitives. Every project reinvents streaming markdown, tool-call panels, diff rendering, confirmation prompts, and slash commands from scratch.

The reference UX — Claude Code, Codex, Aider, Toad — converges on a recognizable shape: linear scrollback, streaming markdown that updates in place without flicker, collapsible tool calls, unified diffs, slash commands, an `@`-file picker. **In Python, no library delivers this shape as a coherent primitive.**

Other ecosystems have closed this gap:

- TypeScript: `pi-tui` (Mario Zechner), powering Mastra Code and others
- TypeScript: `claude-code-kit`, the extracted Claude Code components
- Go: Bubble Tea + Lipgloss, the Charm stack
- Rust: Ratatui

Python has no equivalent. The bet of this PRD is that the gap is real, the timing is right, and the scope is tractable.

## 2. Goals

1. Deliver a small, opinionated Python library for building coding-agent terminal UIs.
2. Optimize for the **linear-scrollback** UX model (no alt-buffer for the conversation).
3. Provide first-class primitives for the patterns every coding agent needs: streaming markdown, tool calls, diffs, confirmations, slash commands, file pickers.
4. Stay framework-agnostic at the agent layer — works with LangGraph, smol-agents, raw OpenAI/Anthropic SDK calls, or custom loops.

5. Match `pi-tui`-class rendering quality (zero flicker on modern terminals via synchronized output).
6. Keep the API small enough that an agent loop fits in under 100 lines of user code.

### 3. Non-Goals

- **Not a general TUI framework.** Textual already fills that slot. Multi-pane dashboards, complex layouts, mouse-driven widgets are out of scope.
- **Not an agent framework.** No LLM client, no tool dispatcher, no memory store, no orchestration. The library renders; the user drives.
- **Not alt-buffer / full-screen mode.** The whole point is to preserve native scrollback and copy-paste. If you need a fixed-region UI, use Textual.
- **Not Windows-first.** Best-effort Windows Terminal support, but the development target is Unix terminals (iTerm2, Ghostty, Kitty, Alacritty, Wezterm, tmux).
- **No remote/web rendering** in v1. Toad's ACP-style decoupling is interesting but a separate initiative.

### 4. Target Users

**Primary:** Python developers building coding agents — researchers, indie devs, internal tooling teams — who want Claude-Code-quality UX without writing a renderer from scratch.

**Secondary:** Agent framework authors (LangGraph, smol-agents) who want a recommended terminal frontend.

**Tertiary:** Anyone building a chat-shaped Python CLI tool that happens to need streaming markdown and structured output blocks (data analysis assistants, RAG explorers, etc.).

### 5. User Stories

1. *As an agent developer*, I want to render streaming markdown from an LLM in place, without screen flicker, so the user sees a smooth typing experience.
2. *As an agent developer*, I want to display a tool call as a structured, collapsible block with name, arguments, and result, so the user can scan execution at a glance.
3. *As an agent developer*, I want to render a unified diff for a file edit and prompt the user to approve, reject, or edit it, so destructive actions stay user-controlled.
4. *As an agent developer*, I want slash commands (`/clear`, `/undo`, `/model`) and `@`-mention file completion in the input line, with no extra wiring.
5. *As an end user*, I want native terminal copy-paste, tmux scrollback, and Cmd-F search to keep working — the agent should feel like part of my shell, not a separate app that hijacks the screen.

6. *As an end user*, I want to scroll back through prior turns and have the rendering remain stable (no rewrites of historical output when new content streams in).

## 6. Functional Requirements

### 6.1 Output primitives

- **Streaming markdown renderer.** Accepts an async iterator of string chunks; updates the active region in place using synchronized output. Final state commits to scrollbar as immutable rendered text.
- **Tool call block.** Structured rendering with name, arguments (JSON-pretty), status (pending / running / done / error), and result. Collapsible by default after completion.
- **Diff block.** Unified diff with syntax-highlighted content, line numbers, and per-hunk markers. Optional side-by-side mode.
- **Code block.** Syntax-highlighted via Rich's existing Pygments integration.
- **Status line / spinner.** Transient status indicator during long operations, removed cleanly on completion.
- **Panel / box.** Generic bordered container for arbitrary content, with title and optional dim/dim-on-complete styling.

### 6.2 Input primitives

- **Multi-line input** with bracketed paste, history (persisted to disk), reverse-i-search.
- **Slash command framework.** Register handlers, get autocomplete and help-text rendering for free.
- **@-file picker.** Triggered by `@`, fuzzy-matches against the current directory, respects `.gitignore`.
- **Confirmation prompts.** Yes/no, multi-choice, free-text-with-default — all rendered in scrollbar, all keyboard-driven.
- **Interrupt handling.** Ctrl-C cancels the in-flight stream cleanly without corrupting terminal state.

### 6.3 Loop integration

- **Async-native API**, with sync wrappers for callers who don't want asyncio.
- **Pluggable message model.** Library does not impose OpenAI vs Anthropic message shapes; user maps their domain types to render calls.
- **Event hooks** for: turn started, turn completed, tool call started, tool call completed, user input received, interrupt received.

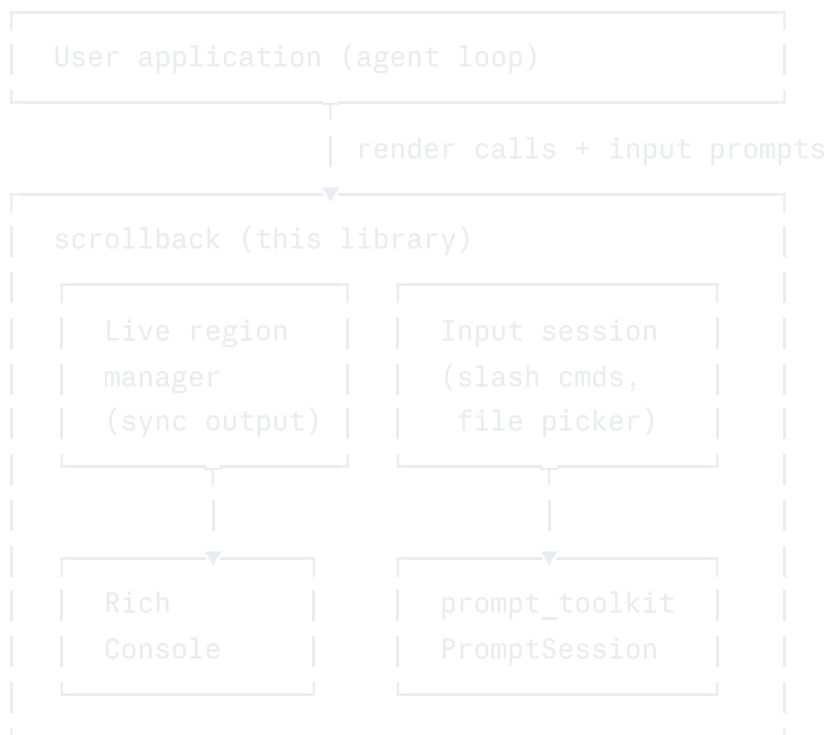
## 7. Non-Functional Requirements

- **Render latency:**  $\leq 16\text{ms}$  per chunk during streaming on a reference terminal (Ghostty on M-series Mac).
- **Zero flicker** on terminals supporting synchronized output (CSI ?2026h/l). Graceful degradation otherwise.
- **Cold start:** library import + first render  $\leq 100\text{ms}$ .
- **Dependencies:** Rich, prompt\_toolkit. Nothing else in the core. Optional extras (e.g., tree-sitter for advanced syntax) gated behind install extras.
- **Type safety:** full type hints, `py.typed`, mypy strict mode passing.
- **Python:** 3.10+.
- **Testing:** snapshot tests on the rendered ANSI output for every primitive, replayed against a recorded terminal for cross-emulator validation.
- **License:** MIT. Goal is broad adoption; no patent or copyleft barriers.

## 8. Technical Approach

### 8.1 Architectural thesis

The library is a thin orchestration layer over Rich (rendering) and prompt\_toolkit (input), with one critical addition: a **scrollback-aware live region manager** that uses synchronized output to update the in-flight turn without flickering or polluting scrollback history.



## 8.2 Live region model

The conversation lives in normal scrollback. Only the **currently-streaming turn** uses a managed region, wrapped in synchronized output sequences. When the turn completes, the region is "committed" — the synchronized wrapper is removed, the content becomes ordinary terminal scrollback, and subsequent renders never rewrite it.

This is the inverse of Textual's model and the same model `pi-tui` and Claude Code use. It's why scroll, search, and selection keep working.

## 8.3 What we borrow vs build

### Borrow directly:

- Rich's segment-based rendering, markdown parser, syntax highlighting, panels.
- `prompt_toolkit`'s input handling, history, key bindings, completion infrastructure.
- `pi-tui`'s synchronized-output rendering pattern (port the model, not the code).

### Build from scratch:

- Live region manager (~300-500 LOC).
- Tool call / diff / confirmation primitives (composed from Rich panels).
- Slash command framework and `@`-file picker (thin layer over `prompt_toolkit` completers).
- Snapshot test harness for ANSI output.

## 8.4 API sketch

```
python
```

```

from rollback import Session, ToolCall, Diff

async def main():
    async with Session() as ui:
        while True:
            user_input = await ui.prompt("> ")
            if user_input.startswith("/"):
                await ui.handle_command(user_input)
                continue

            async with ui.assistant_turn() as turn:
                async for chunk in agent.stream(user_input):
                    if chunk.kind == "text":
                        await turn.append_markdown(chunk.text)
                    elif chunk.kind == "tool_call":
                        async with turn.tool_call(chunk.name, chunk.args) as tc:
                            result = await execute_tool(chunk)
                            await tc.complete(result)
                    elif chunk.kind == "diff":
                        approved = await turn.diff(chunk.path, chunk.patch).confirm()
                        if approved:
                            apply_patch(chunk.path, chunk.patch)

```

The shape is: sessions own input/output; turns own a single agent response; primitives (tool call, diff, confirmation) are async context managers that handle their own lifecycle and rendering.

## 9. Milestones

Version	Scope	Estimate
v0.1	Live region manager, streaming markdown, basic input loop	2–3 weeks
v0.2	Tool call blocks, diff blocks, confirmation prompts	2–3 weeks
v0.3	Slash commands, <code>@</code> -file picker, history, interrupt handling	2 weeks
v0.4	Snapshot test harness, cross-terminal validation matrix	2 weeks
v0.5	Documentation, three example agents (LangGraph, smol-agents, raw SDK)	2–3 weeks
v1.0	API freeze, polish, public release	2–4 weeks

**Total to v1.0:** ~3–4 months at full focus.

## 10. Success Metrics

### Adoption (6 months post-v1.0):

- $\geq 10$  distinct projects on GitHub depending on the library
- $\geq 1$  agent framework recommends it as a frontend
- PyPI downloads trending positively

### Quality:

- Snapshot test suite covers every primitive
- Zero flicker on the validation terminal matrix (Ghostty, iTerm2, Kitty, Alacritty, Wezterm, tmux+xterm, Windows Terminal)
- Issue tracker dominated by feature requests, not rendering bugs

### Developer experience:

- "Hello world" agent loop  $\leq 30$  lines
- Full-featured agent loop (tools, diffs, slash commands)  $\leq 100$  lines
- Time-to-first-render  $\leq 100\text{ms}$

## 11. Risks & Mitigations

Risk	Likelihood	Impact	Mitigation
Terminal compatibility long tail (synchronized output, Unicode width, etc.)	High	High	Cross-terminal test matrix from week 1; graceful degradation as default
Rich's API constraints force workarounds	Medium	Medium	Prototype the live region against Rich early; fork or replace specific Rich components if blocked
Scope creep into full TUI framework	High	High	Hard non-goals in this doc; reject every "what about multi-pane" request
Async/sync impedance with user agent loops	Medium	Low	Provide thin sync wrappers; document async as primary
Someone else ships a competing library mid-build	Medium	Medium	Move fast to v0.1; the scrollbar-first opinion is differentiated even if a generic option appears
The thesis is wrong — Python users prefer Textual or Streamlit	Low	High	Validate with 3–5 prospective users between v0.1 and v0.2 before further

Risk	Likelihood	Impact	Mitigation
			investment

## 12. Open Questions

1. **Name.** `scrollback` is descriptive but generic and likely conflicts on PyPI. Candidates: `agenttty`, `livescroll`, `agentline`, `terminus`, `streamline`, `tty-agent`. Decide before v0.1 release.
2. **License.** MIT is the default for adoption. Worth discussing whether Apache 2.0 (patent grant) is preferred for any enterprise users we want to court.
3. **Should the library own the agent loop, or stay purely a renderer the user drives?** Current PRD says renderer-only. Worth pressure-testing once we have v0.2 in hand.
4. **How opinionated about message structure?** Library currently stays neutral. If the cost is too much per-project boilerplate, may add an optional "OpenAI-shaped" or "Anthropic-shaped" adapter.
5. **Is image rendering (Kitty / iTerm2 inline image protocol) v1 or v2?** Lean v2, but worth a stub interface in v1 so it can land later without breaking changes.
6. **Open source from day one, or build privately to v0.5 then release?** Releasing early gets feedback but invites bikeshedding before the API is stable.
7. **How much does this overlap with Textual's roadmap?** Worth a conversation with Will McGugan — Toad demonstrates Textualize is thinking about agent UX too. Collaboration vs competition is a real choice.

## 13. Appendix: References

- `pi-tui` (TypeScript): <https://github.com/badlogic/pi-mono> — reference for linear-scrollback rendering model
- Toad (Python/Textual): demonstrates polish ceiling for agent TUIs in Python, alt-buffer model
- `claude-code-kit` (TypeScript): reference for ANSI parser and component decomposition
- Aider (`aider/io.py`, `aider/main.py`): cleanest Python example of prompt\_toolkit + Rich agent loop in production
- Rich, prompt\_toolkit documentation
- Synchronized output spec:  
<https://gist.github.com/christianparpart/d8a62cc1ab659194337d73e399004036>