Binoculars VS Code Extension

This directory contains a working VS Code extension front end that reuses Binoculars scoring/rewrite logic through a persistent Python bridge.

For complete end-user workflows and troubleshooting, see:

- `../USERGUIDE-VC.md`

Current Scope

Implemented:

- Activity Bar container (`Binoculars`) with `media/binoculars-icon.png`
- Controls view in Activity Bar with one-click commands and live status
- Persistent backend bridge process (`python/binoculars_bridge.py`)
- Extension commands + default hotkeys
- Settings-driven runtime configuration (Python path, bridge path, config path, GGUF overrides, rewrite LLM config path)
- Analysis rendering in editor:
  - LOW/HIGH text colorization
  - Unscored-region rendering
  - Per-line contribution gutter bars
- Chunk workflows:
  - Analyze full document
  - Analyze next chunk
- Rewrite workflow:
  - Rewrite selection or active red contributor under cursor
  - QuickPick chooser with approximate B impact

Intentionally out of scope here:

- Markdown preview pane integration
- Synonym functionality (already covered by existing VS Code ecosystem)
- Spellchecking integration

Directory Layout

- `package.json`: extension manifest (commands, keybindings, settings)
- `src/extension.ts`: VS Code UI logic + decorations + command handlers
- `src/backendClient.ts`: persistent bridge client (JSON over stdio)
- `python/binoculars_bridge.py`: backend API adaptor over core `binoculars.py`
- `schemas/bridge.protocol.schema.json`: protocol schema for request/response envelopes
- `media/binoculars-icon.png`: Activity Bar icon

Commands / Hotkeys

Default bindings (users can remap via Keyboard Shortcuts):

- `Binoculars: Analyze Chunk` -> `Ctrl+Alt+B`
- `Binoculars: Analyze Next Chunk` -> `Ctrl+Alt+N`
- `Binoculars: Rewrite Selection` -> `Ctrl+Alt+R`
- `Binoculars: Clear Priors` -> `Ctrl+Alt+C`

Settings

Key settings contributed by this extension:

- `binoculars.backend.pythonPath`
- `binoculars.backend.bridgeScriptPath`
- `binoculars.configPath`
- `binoculars.textMaxTokensOverride`
- `binoculars.topK`
- `binoculars.models.observerGgufPath`
- `binoculars.models.performerGgufPath`
- `binoculars.externalLlm.*`
- `binoculars.render.*`

Notes:

- Current defaults point to the existing local paths used in this repository.
- All paths remain configurable in VS Code settings.
- GGUF overrides are applied by bridge-generated runtime config patching.
- External rewrite LLM config path is passed via `BINOCULARS_REWRITE_LLM_CONFIG_PATH`.

Theme Mode

This implementation is tuned for dark mode first.

- Dark palette is primary for LOW/HIGH/unscored overlays and gutter bars.
- A conservative light-theme palette fallback is included, but full light-mode tuning is a follow-up task.

Build / Run (from this folder)

```
npm install
npm run compile
```

Then run the extension in VS Code Extension Development Host (F5) from this folder.

Shared-Backend Direction (Tk + VS Code)

Yes, it makes architectural sense to refactor the Tk front end to consume the same persistent bridge API.

Why:

- Single source of truth for scoring/rewrite/chunk semantics.
- Faster feature parity between GUI clients.
- Lower regression risk (one API contract, one test harness).
- Easier future front ends (web/TUI) without duplicating model lifecycle code.

Recommended migration path:

1. Keep current Tk code functional.
2. Route Tk Analyze/AnalyzeNext/Rewrite calls through bridge methods behind a feature flag.
3. Move chunk-state logic and rewrite approximation baseline logic into bridge responses.
4. Retire duplicated Tk-only scoring paths after parity tests pass.