Binoculars Project Agent Guide

1) Project Purpose

This repository implements a local, likelihood-based AI text forensics tool inspired by Binoculars.

Primary goal:

- Score markdown text using two related llama.cpp models (observer + performer) and compute:
  - `logPPL` for observer
  - `logXPPL` cross-entropy term
  - `B = logPPL / logXPPL` (Binoculars ratio)

Why this exists:

- `initial-scoping.md` documents that API-only approaches (OpenAI/Ollama top-N logprobs) are approximate and fragile.
- The chosen direction is faithful local scoring with full logits via `llama-cpp-python`.

2) Current Project State

Status:

- Functional CLI + GUI prototype with regression coverage for the `v1.1.x` line.
- Core scoring is stable; calibration/classification is not implemented.
- GUI supports iterative rewrite workflows with approximate impact scoring and explicit full re-analysis.
- GUI also supports fast synonym-assisted edits, one-level undo for tracked mutations, and transient status messaging that restores metrics automatically.
- VS Code extension is active and featureful:
  - chunk-aware `Analyze Chunk` / `Analyze Next Chunk`
  - rewrite selection/line with ranked options
  - colorization + gutter bars + hover diagnostics
  - `Toggle Colorization` runtime command
  - prior contributor faint backgrounds (major contributors only)
  - sidecar state restore (`<doc>.json`)

Latest known commit at time of this guide update:

- `225b75e`

3) Repository Map

Key files:

- `binoculars.py`: main scoring CLI + GUI application.
- `binoculars.sh`: venv-activating wrapper runnable from any directory.

- `binocular.sh`: alias wrapper (`exec binoculars.sh`).
- `config.binoculars.json`: master profile map (`fast`/`long`) + default profile + optional per-profile `max_tokens` override.
- `config.binoculars.llm.json`: optional OpenAI-compatible rewrite backend config for GUI rewrites.
- `config.llama31.cuda12gb.fast.json`: default profile (`text.max_tokens=4096`).
- `config.llama31.cuda12gb.long.json`: long profile (`text.max_tokens=12288`).
- `README.md`: project overview and CLI/GUI reference.
- `USERGUIDE-GUI.md`: GUI-specific interactive workflow guide.
- `USERGUIDE-VC.md`: VS Code extension workflow guide.
- `vscode-extension/src/extension.ts`: extension UI/decoration/command logic.
- `vscode-extension/src/backendClient.ts`: persistent JSON bridge client.
- `vscode-extension/python/binoculars_bridge.py`: bridge backend process adapter.
- `vscode-extension/package.json`: command/menu/settings manifest.
- `tests/test_regression_v1_1_x.py`: regression checks.
- `tests/fixtures/Athens.md`: stable fixture copy for regression tests.
- `initial-scoping.md`: technical scoping and tuning lessons.

Local assets (present on this machine):

- `models/Meta-Llama-3.1-8B-Q5_K_M-GGUF/meta-llama-3.1-8b-q5_k_m.gguf` (~5.4G)
- `models/Meta-Llama-3.1-8B-Instruct-Q5_K_M-GGUF/meta-llama-3.1-8b-instruct-q5_k_m.gguf` (~5.4G)

4) How the Implementation Works

High-level scoring flow in `binoculars.py`:

1. Load JSON config and validate required sections (`observer`, `performer`).
2. Read input markdown from file or stdin.
3. Tokenize text with each model in `vocab_only=True` mode.
4. Enforce exact tokenization match (hard fail if mismatch).
5. Infer `n_ctx` (auto when configured as `0`).
6. Load observer with `logits_all=True`, run `eval(tokens)`.
7. Compute observer `logPPL`, write observer logits to memmap.
8. Unload observer (VRAM reduction).
9. Load performer with `logits_all=True`, run `eval(tokens)`.
10. Compute performer `logPPL` (informational) and cross `logXPPL` using observer memmap + performer logits.
11. Compute `B = logPPL(observer) / logXPPL(observer, performer)`.
12. Emit text or JSON output.
13. Remove cache unless `cache.keep=true`.

Design choice:

- Models are loaded sequentially (never concurrently) to reduce VRAM pressure.

5) Config Profiles and Intent

`config.llama31.cuda12gb.fast.json`:

- `max_tokens: 4096`
- `offload_kqv: true`
- `n_batch: 1024`

`config.llama31.cuda12gb.long.json`:

- `max_tokens: 12288`
- `offload_kqv: true`
- `n_batch: 1024`

Both shipped profiles:

- Use Llama 3.1 8B base + instruct Q5_K_M sibling models.
- Use `n_ctx: 0` (auto = token count).
- Use cache dtype `float16`.

Master config override behavior:

- `config.binoculars.json` may define each profile as:
  - string path, or
  - object with:
    - `path`: concrete config JSON
    - `max_tokens`: optional non-negative override for `text.max_tokens`

Optional rewrite backend config (`config.binoculars.llm.json`):

- If file missing or disabled: use internal performer rewrite generation.
- If present and reachable: use configured OpenAI-compatible endpoint.
- If present but invalid/unreachable at runtime: auto-fallback to internal generation.

6) Environment and Bootstrap

Observed local state:

- Python in repo venv: `3.10.12`.

Baseline dependencies:

```
venv/bin/pip install numpy llama-cpp-python
```

Optional dependency for richer synonym lookup:

```
```

```
venv/bin/pip install nltk
venv/bin/python - <<'PY'
import nltk
nltk.download('wordnet', quiet=True)
nltk.download('omw-1.4', quiet=True)
PY
```

Example scoring run:

```
venv/bin/python binoculars.py --config fast your_doc.md --json
```

GUI run:

```
venv/bin/python binoculars.py --config fast --gui your_doc.md
```

7) Output Contract

JSON output includes:

- `input` metadata (chars, tokens, transitions)
- `observer` (`logPPL`, `PPL`)
- `performer` (`logPPL`, `PPL`)
- `cross` (`logXPPL`, `XPPL`)
- `binoculars.score` (`B`)
- `cache` details

Important:

- Script returns scores only. No built-in threshold classifier labels.
- GUI focuses on iterative editing + rescoring:
  - `Analyze` computes exact metrics for the active analyzed chunk and refreshes heatmap coverage.
  - rewrites show approximate impact; exact B requires Analyze.
  - status explicitly marks B stale after edits/rewrites.
  - non-analysis status messages are transient and then restore metrics (Undo success is intentionally brief).

GUI rewrite behavior:

- Right-click on red (`LOW`) scored segments to request 3 rewrites.
- Or highlight a block (multi-line) and right-click for block rewrites.
- Selection rewrites:
  - are rounded to full lines

  - are clamped to scored/analyzed text if needed
  - preserve unchanged source lines when model output accidentally omits/collapses them
- Popup supports scrolling and keyboard selection (`1`/`2`/`3`, `Q`/`Esc` to cancel).
- Options are sorted by expected B increase (more human-like first).

GUI synonym behavior:

- Left-click a word in the left pane to trigger synonym lookup after a short debounce.
- Synonym panel (bottom of right pane) shows up to 9 options in 3 columns with buttons `1..9`.
- Source order: local fallback -> WordNet (if installed) -> Datamuse API fallback.
- Applying a synonym is tracked as one undoable mutation.

GUI undo behavior (single level):

- Toolbar has `Undo` button.
- Supported tracked operations:
  - selected-block delete (`Delete` or `Backspace`),
  - synonym replacement,
  - red-segment rewrite replacement,
  - highlighted-block rewrite replacement.
- Undo is invalidated if document text changes after the tracked operation.

GUI identity details:

- Window/app name is set to `Binoculars` (including Linux WM class/appname hints).
- GUI icon is drawn in code (owl with large eyes) via Tk `PhotoImage`.

VS Code extension behavior (current):

- Commands:
  - `Binoculars: Analyze Chunk`
  - `Binoculars: Analyze Next Chunk`
  - `Binoculars: Rewrite Selection`
  - `Binoculars: Clear Priors`
  - `Binoculars: Toggle Colorization`
  - `Binoculars: Restart Backend`
- Color model:
  - Major LOW/HIGH contributors are colorized.
  - Minor contributors are rendered neutral (`light gray` in dark theme, `black` in light theme).
  - Gutter bars remain available independently of text colorization.
- Prior overlays:
  - After re-analysis, prior contributor overlays are captured as faint backgrounds.
  - Prior overlays are restricted to prior major contributors (top-`k` LOW/HIGH), not minor rows.
  - `Clear Priors` clears prior overlays without deleting current analysis state.
- Toggle colorization:
  - `Toggle Colorization` hides/shows text overlays at runtime.

    - Re-enabling restores overlays from in-memory state (including prior/edited backgrounds when present).
- Persistence:
    - Sidecar save/load includes chunk state, edited ranges, rewrite ranges, and prior overlay ranges.
    - `priorChunkB` remains in-session and is intentionally not restored from persisted state.

8) Lessons Learned / Gotchas

1. Full-logit requirement is non-negotiable for faithful Binoculars.

- API top-k logprob approaches are approximate and biased.

2. Tokenizer/vocab alignment is critical.

- Script hard-fails on observer/performer tokenization mismatch.

3. Memory pressure is dominated by `(tokens * vocab)` with `logits_all=True`.

- `text.max_tokens` is the primary safety valve.

4. `n_ctx: 0` auto-sizing avoids over-allocation and many avoidable failures.

5. GUI rewrite scoring is intentionally approximate between analyses.

- Approximate option ranking is local observer-logPPL based.
- Cross term is not recomputed until Analyze.

6. Markdown is treated as text.

- Rendering is for convenience; scoring is raw text-token based.

9) Chunk-Aware GUI Analysis (Implemented Behaviour)

Chunk-aware large-file analysis is implemented in the GUI.

9.1) Analyze and Analyze Next Semantics

1. First `Analyze`:

- Starts at document char `0`.
- Scores forward until token/memory limit for that run.

2. `Analyze Next`:

- Starts at contiguous covered tail (`analysis_covered_until`).
- Scores the next token-limited chunk.
- Remains available until contiguous coverage reaches end-of-document.

3. Later `Analyze` runs:

- Resolve active chunk.
- Start scoring at `active_chunk.char_start` (not cursor char).
- Recompute that chunk from its start boundary.

9.2) Active Chunk Resolver Priority

Resolver order is deterministic:

1. Current selection overlap with analyzed chunks (largest overlap wins).
2. Else chunk containing visible insert/cursor line.
3. Else chunk with largest overlap against visible line window.
4. Else nearest analyzed chunk by char distance.

Status metrics and rewrite approximation baselines use this active chunk.

9.3) Chunk Boundary Mutability (Important)

Chunks are not immutable “mini-documents”.

- Chunk metrics are stored per chunk descriptor.
- Coverage intervals are merged for rendering and unscored complements.
- On overlap, old descriptor(s) are replaced by the newest re-analysis descriptor.
- Therefore chunk end boundaries may move after edits.

Example:

- First pass may produce chunk 1 covering lines `1-999`.
- User at line `999` presses `Analyze`.
- Analyzer restarts at chunk-1 start (line 1), not line 999.
- After edits, same run may now end at line `972` or `1031` due to token-density changes.

9.4) Rewrite Approximation in Chunk Context

- Rewrite requests target selected span/segment as usual.
- Baseline metrics are resolved from the request/active chunk.
- Approximate scoring context is clamped to chunk bounds when available.

9.5) Rendering Model

- Combined annotations from all chunk profiles are rendered globally.
- Unscored intervals are computed as complement of merged scored coverage.
- Left gutter bars and preview backgrounds reflect multi-chunk state.

10) Known Gaps / Next Development Priorities

Priority backlog:

1. Add calibration pipeline:

- dataset runner + threshold selection + FPR/TPR reporting.

2. Add sliding-window scoring:

- support very long docs without full-doc logits materialization.

3. Add tests:

- synthetic math checks for perplexity/cross-perplexity and rewrite post-processing guards.

4. Add dependency pinning:

- `requirements.txt` or `pyproject.toml`.

5. Add reproducible benchmark script:

- throughput/memory across profiles and input lengths.

11) Agent Operating Notes

When resuming work:

1. Verify environment first:

- dependencies, model paths, writable cache directory.

2. Preserve core math semantics unless intentionally changing them.

- Any changes to token alignment, cross-entropy math, or truncation behavior must be explicit and documented.

3. Preserve sequential model loading unless redesign is intentional and benchmarked.

4. For GUI rewrite changes, keep user control explicit.

- Do not auto-run full Analyze after each rewrite.
- Keep approximate impact messaging clear.

5. If classification labels are added in future:

- keep raw numeric outputs and expose calibration metadata.

12) Non-Goals (Current)

Not currently in scope:

- Definitive authorship claims.
- Remote API-only detector approximations for scoring core.
- Production web service deployment.

13) VS Code Marketplace Roadmap

Goal:

- Publish `vscode-extension` to the VS Code Marketplace in a way that non-technical users can install and run it successfully.

13.1) Publishing Prerequisites

1. Publisher setup:

- Create/verify Azure DevOps publisher identity for Marketplace.
- Generate Personal Access Token (PAT) with Marketplace publish permissions.
- Add secure local/CI publishing flow (`vsce publish` or GitHub Action).

2. Manifest hardening (`vscode-extension/package.json`):

- Add `repository` field (currently missing warning during packaging).
- Keep `displayName`, `description`, `categories`, icon, and command titles user-friendly.
- Ensure license path is valid for packaged extension.

3. Release hygiene:

- Add `CHANGELOG.md`.
- Define semantic versioning and release notes process.
- Add automated prepublish checks (`npm run compile`, smoke checks).

13.2) Installation Friction Assessment (Config Streamlining)

Short answer:

- Yes, streamlining is likely needed for broad adoption.
- For expert users, current explicit paths can work; for general users, manual path/model setup is too error-prone.

Why:

- Current defaults point to machine-specific absolute paths.
- Users must have Python, dependencies, config JSON, and GGUF model paths aligned.
- Without guided setup, first-run failure rate will be high outside the current dev environment.

13.3) Accessibility Roadmap (Broader Audience)

Phase 1 (minimum for Marketplace launch):

1. Replace machine-specific defaults with portable defaults:

- Use `${workspaceFolder}` where appropriate.
- Leave optional paths empty when unknown and detect at runtime.

2. Add first-run preflight + guided setup command:

- Validate Python executable, bridge script, config file, model files.
- Show actionable one-click fixes/open-settings shortcuts.
- Persist discovered valid paths automatically.

3. Improve error UX:

- Human-readable diagnostics in notifications and output channel.
- Clear distinction between missing dependency, missing model, bad config, and backend startup failure.

Phase 2 (recommended post-launch):

1. Setup wizard:

- Multi-step onboarding UI for selecting profile/config/models.
- "Test backend" button before first analyze.

2. Optional quickstart profile:

- Ship a template config and docs that minimize manual edits.
- Provide explicit "local-only" and "external rewrite backend" setup paths.

3. Telemetry-free health metrics (local only):

- Count setup failures in session and surface targeted help (no remote telemetry required).

Phase 3 (best usability):

1. Dependency/bootstrap helper:

- Command to create/check venv and install Python dependencies.
- Optional model path validator/downloader integration (if licensing/distribution allows).

2. Cross-platform QA matrix:

- Linux/macOS/Windows first-run validation with clean machines.

13.4) Acceptance Criteria Before Public Marketplace Push

1. Fresh-machine install succeeds with guided steps (no code edits required).
2. User can run first `Analyze Chunk` within a short onboarding flow.
3. Common failures provide direct remediation links/commands.
4. Documentation is aligned:

- Root `README.md`
- `vscode-extension/README.md`
- `USERGUIDE-VC.md`