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.

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.
- `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 document metrics and refreshes heatmap.
  - 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`.

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) Known Gaps / Next Development Priorities

Priority backlog:

1. Implement chunk-aware GUI analysis for very large files (top priority).

9.1) Chunk-Aware GUI Analysis: Functional Requirements

1. `Analyze` behavior:

- On first run, analyze the first analyzable chunk.
- On subsequent runs, analyze the current active chunk (not always chunk 1).

2. `Analyze Next` behavior:

- Button appears only after first successful chunk analysis when unscored text remains.
- Each click analyzes the next uncovered chunk after the highest covered tail.
- Button remains visible until full document coverage is complete.

3. Chunk-aware status readout:

- Status-bar `B` and other metrics show values for the active chunk.
- Active chunk selection priority:
  - chunk containing currently selected text block,
  - else chunk containing current insert/cursor line,
  - else chunk containing majority of visible editor lines when selection/cursor anchor is off-screen.

4. Rewrite behavior:

- Right-click rewrite requests still target requested segment/block.
- Approximate B-impact for options must use the active/requested chunk baseline metrics.

5. Visuals:

- Heatmap/line bars should show all analyzed chunks.
- Unscored styling must apply to every not-yet-analyzed region (not just trailing tail).

9.2) Recommended GUI State Model

Add/maintain these GUI state fields in `launch_gui`:

- `analysis_chunks`: list of chunk descriptors sorted by `char_start`.
- `active_chunk_id`: stable identifier or list index for current active chunk.
- `analysis_next_available`: bool.
- `analysis_covered_until`: highest contiguous covered char from document start.
- `analysis_last_line_by_chunk`: optional map chunk_id -> last scored line.

Recommended chunk descriptor shape:

```
{
  "id": int,
  "char_start": int,
  "char_end": int,
  "transitions": int,
  "metrics": {
    "binoculars_score": float,
    "observer_logPPL": float,
    "performer_logPPL": float,
    "cross_logXPPL": float,
  },
  "profile": dict,   # paragraph profile adjusted to document-global char
offsets
}
```

9.3) Chunk Construction Strategy

Use existing scoring engine, but analyze text slices:

1. Determine target chunk start char:

- first Analyze: `0`
- Analyze Next: first uncovered char after `analysis_covered_until`
- Analyze active chunk: `active_chunk.char_start`

2. Build chunk text slice:

- Start from chosen char start.
- Respect paragraph boundaries where practical (avoid splitting mid-line if easy).
- Let current `text.max_tokens` cap define maximum analyzable chunk size.

3. Run existing analyzer on slice and shift offsets:

- Call existing `analyze_text_document` on chunk text.
- Shift returned paragraph row `char_start/char_end` by chunk `char_start` so GUI can render globally.

9.4) Toolbar/UI Changes

Add `Analyze Next` button in toolbar:

- Hidden/disabled initially.
- Show/enable after first chunk analysis if document has uncovered text.
- Disable while analyzing/rewrite-busy like other controls.
- Hide or disable permanently when full coverage reached.

9.5) Active Chunk Resolver (Deterministic)

Implement a pure helper with explicit priority:

1. If `sel` exists and overlaps one or more analyzed chunks, choose chunk with maximum overlap.
2. Else if insert line is visible and belongs to an analyzed chunk, choose it.
3. Else compute visible line range via `@0,0` and bottom index; choose analyzed chunk with maximum visible-line overlap.
4. If no analyzed chunk matches, fallback to nearest by char distance.

Keep this helper reusable from:

- status refresh
- Analyze button handler
- rewrite request handlers

9.6) Analyze/Analyze Next Execution Flow

1. Start worker with chosen chunk slice.
2. On success:

- merge/replace chunk descriptor in `analysis_chunks` by overlap rules.
- recompute covered intervals and contiguous `analysis_covered_until`.
- recompute active chunk.
- recompute status line from active chunk metrics.

3. Preserve existing edit/prior behavior:

- keep stale suffix logic unchanged.
- do not auto-trigger whole-document recomputation.

9.7) Rendering + Unscored Regions

Current renderer assumes one profile/tail. Replace with interval-based rendering:

1. Merge analyzed chunk intervals.
2. Apply segment tags for all chunk profiles.
3. Apply `unscored` tag to complement intervals across full text.
4. Rebuild line contribution map from combined rows/annotations from all analyzed chunks.

9.8) Rewrite Approximation in Chunk Context

When user requests rewrite:

1. Resolve request chunk by overlap with span selection.
2. Use chunk metrics (`B`, observer logPPL, cross logXPPL, transitions) as baseline for approximate scoring.
3. Limit local approximation text context to chunk bounds (or clamp hard to chunk bounds) to avoid cross-chunk baseline mismatch.
4. Preserve current safeguards for omitted lines/trailing newlines.

9.9) Stepwise Implementation Plan

Implement in this order to reduce risk:

1. Add chunk state model + helpers (interval merge, active chunk resolver).
2. Add `Analyze Next` button and visibility/state wiring.
3. Refactor analysis success path to store/update chunk descriptors.
4. Refactor rendering to support multi-chunk annotations + unscored complements.
5. Switch status line source from singleton metrics to active chunk metrics.
6. Wire rewrite handlers to chunk-aware baseline metrics.
7. Add tests and update docs (`README.md`, `USERGUIDE-GUI.md`).

9.10) Test Plan (Minimum)

1. Unit tests for interval utilities:

- merge chunk intervals,
- complement/unscored intervals,
- overlap-based chunk selection.

2. Unit tests for active chunk resolver priority:

- selection overlap wins,
- then visible insert line,
- then majority visible window.

3. Regression tests:

- `Analyze Next` appears/disappears correctly.
- status line changes when cursor/selection moves between chunks.
- rewrite approximation uses active chunk baseline.

4. Manual GUI checks:

- very large file with 3+ chunks,
- interleaved edits + Analyze + Analyze Next + rewrites,
- Clear Priors still works and does not alter chunk coverage.

9.11) Definition of Done for This Feature

Feature is done when:

- Large docs can be fully covered via Analyze + Analyze Next without truncating workflow to one chunk.
- Status metrics and rewrite approximations are chunk-correct and predictable.
- UI clearly indicates scored/unscored regions across the full file.
- Existing small-document single-pass behavior remains unchanged.

2. Add calibration pipeline:

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

3. Add sliding-window scoring:

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

4. Add tests:

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

5. Add dependency pinning:

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

6. Add reproducible benchmark script:

- throughput/memory across profiles and input lengths.

10) 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.

11) Non-Goals (Current)

Not currently in scope:

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