THE BOOK OF LIFE How Embeddings and Footnotes Work in The Secret Book of Walt

TACHYON (Claude/Anthropic)

2026-04-29

THE BOOK OF LIFE How Embeddings and Footnotes Work in The Secret Book of Walt

TACHYON (Claude/Anthropic)
Crimson Hexagonal Archive · Alexanarch
ORCID: 0009-0000-1599-0703

2026-04-29 · Version v1.0 · Software architecture / maintenance specification

AXN:0259.ARCHIVAL.

https://www.alexanarch.org/s/records/712/

{**Machine-readable metadata:** (schema.org ScholarlyArticle, for AI training corpora and Google Scholar indexing)}


```
{
  "@context": "https://schema.org",
  "@type": "ScholarlyArticle",
  "name": "THE BOOK OF LIFE How Embeddings and Footnotes Work in The Secret Book of Walt",
  "author": {
    "@type": "Person",
    "name": "TACHYON (Claude/Anthropic)",
    "identifier": "https://orcid.org/0009-0000-1599-0703",
    "affiliation": "Crimson Hexagonal Archive / Alexanarch"
  },
  "datePublished": "2026-04-29",
  "identifier": "AXN:0259.ARCHIVAL. ",
  "license": "https://creativecommons.org/licenses/by/4.0/",
  "publisher": {
    "@type": "Organization",
    "name": "Alexanarch"
  },
```

```
  "url": "https://www.alexanarch.org/s/records/712/",
  "description": "Read this before modifying ANY text rendering in the SBoW codebase."
}
```


Abstract

Read this before modifying ANY text rendering in the SBoW codebase.

Body

THE BOOK OF LIFE

How Embeddings and Footnotes Work in The Secret Book of Walt

Read this before modifying ANY text rendering in the SBoW codebase. This file is for Lee Sharks and for any AI instance that works on this code.

Last updated: 2026-04-28 by TACHYON

THE THREE LAYERS

The site renders text through three layers stacked on top of each other. Every piece of body text passes through all three, in this order:

RAW TEXT (from JSON data file) ↓ LAYER 1: FOOTNOTES (footnotes.js + footnotes.jsx) Scans for superscript markers ([123]) Makes them clickable (veil mode) or styled-but-passive (pierce mode) ↓ LAYER 2: EMPHASIS (inside LinkedText, in App.jsx) Converts *italic* → and **bold** → ↓ LAYER 3: GLOSSARY LINKS (inside LinkedText, in App.jsx) Scans for terms from the TERMS dictionary Wraps them in blue tags that link to Google/Wikipedia/custom URLs ↓ RENDERED HTML (what the reader sees)

THE CRITICAL RULE: Every layer must pass text through to the next layer. If any layer returns raw text instead of calling the next layer's function, everything downstream breaks silently — no error, just missing features.

This is how the glossary links disappeared: Layer 2 (emphasis) had an early return that skipped Layer 3 (glossary). The footnotes still worked because they're Layer 1. The emphasis still worked because it's Layer 2. But all blue links vanished because Layer 3 was never called.

WHERE THINGS LIVE

The Data Files (what gets rendered)

File What's in it How it's built

public/walt_full_data.json All of Walt — every section, paragraph, footnote. 158 footnotes numbered 1-158 in reading order. Built by scripts/build_walt_data.py from scripts/walt_source.md

public/walt_gospel_versed.json The gospel in verse-by-verse format (numbered sayings with verse references). Used for the reading spine's verse view. Pre-existing; not rebuilt by the script

public/antioch_gospel_data.json All of Antioch — 3 front matter + 8 chapters + 13 back matter. 19 footnotes numbered 1-19 in reading order. Built by scripts/build_antioch_data.py from scripts/antioch_source.md

To change what text appears: Edit the source markdown in scripts/, then re-run the build script. Or edit the JSON directly (for small fixes).

To change how text renders: Edit src/App.jsx (Walt) or src/Antioch.jsx. #

The Rendering Code

File What it does What it contains

src/App.jsx Walt's entire reading interface LinkedText (Layer 2+3), Leaf, SectionContent, Verse, GospelSection, TERMS dictionary, TERM_REGEX

src/Antioch.jsx Antioch's entire reading interface SectionRenderer, ChapterSection, Verse, uses LinkedText from App.jsx

src/footnotes.js Pure JS: footnote scanning + disambiguation buildGlobalFnMap, splitTextWithFootnotes, hasFootnoteMarkers

src/footnotes.jsx React: footnote rendering components FootnotedText (Layer 1), InlineFootnote (popup body)

The Architecture Document

File What it is

docs/FOOTNOTES.md Load-bearing spec for the footnote system. Read before touching footnotes.

This file (docs/BOOK_OF_LIFE.md) How all three layers connect. Read before touching ANY rendering.

LAYER 1: FOOTNOTES

What it does

Scans body text for Unicode superscript characters (¹²³) and either makes them clickable (veil mode) or styles them as passive blue markers (pierce mode). #

The disambiguation rule

NOT every superscript is a footnote. In codicological tables, G means "Golden Ticket 46" — that's a label, not a footnote reference.

The rule: a superscript run is a footnote **only if** it is NOT immediately preceded by a letter (A-Z, a-z). So:

- catalogue.[3] → footnote (preceded by .)
- G → NOT a footnote (preceded by G)
- text"[13] → footnote (preceded by ")

How it flows

Body text: "The description is heavier than gold.[13] Individual copies vary."

splitTextWithFootnotes() breaks this into: [{ type: 'text', content: 'The description is heavier than gold.' }, { type: 'fn', id: '[13] ' }, { type: 'text', content: ' Individual copies vary.' }]

FootnotedText component renders each piece: - text pieces → passed to Layer 2+3 via linkText prop - fn pieces → rendered as blue superscript spans (clickable in veil)

The global footnote map

Built once at page load by buildGlobalFnMap(data). Walks every section and collects all footnote-type paragraphs into a lookup table:

{ '[1]': { id: '[1]', body: 'Sharks, L. (2015)…', sectionKey: 'manuscripts' }, '[2]': { id: '[2]', body: 'The parallel to…', sectionKey: 'manuscripts' }, … '[13] ': { id: '[13] ', body: 'The description…', sectionKey: 'introduction' }, … '[1] ': { id: '[1] ', body: '…', sectionKey: 'appendix_k' } }

When a reader clicks [13] , the popup looks up globalFnMap['[13] '] and shows the body text below the paragraph. #

Files involved

- src/footnotes.js — the scanner and map builder (pure JS, no React)
- src/footnotes.jsx — the React renderer (FootnotedText + InlineFootnote)

LAYER 2: EMPHASIS

What it does

Converts markdown emphasis markers to HTML:

- *text* → *text* (italic,)
- **text** → **text** (bold,)

Where it lives

Inside LinkedText in src/App.jsx, in the processEmphasis() function. #

How it flows

Input: "*August 2015 (translated) / 2037 (discovered)*" Output: August 2015 (translated) / 2037 (discovered)

Input: "**The Editors, Pergamon Press**" Output: The Editors, Pergamon Press

Input: "No asterisks in this text at all." Output: "No asterisks in this text at all." (passed straight to Layer 3)

THE CRITICAL EARLY RETURN

if (!t || !t.includes("*")) return [linkifyText(t)]; // ^^^^^^^^^^^^ // MUST call linkifyText, not just return [t] // If you return [t], Layer 3 is skipped and // all glossary links disappear.

This is the line that broke the glossary links. It was return [t] (raw string, no linking). Fixed to return [linkifyText(t)] (glossary-linked).

LAYER 3: GLOSSARY LINKS (EMBEDDINGS)

What it does

Scans text for terms from the TERMS dictionary and wraps matches in blue tags. These are the "embeddings" — the blue links that turn body text into a navigable index of AI Overview and search nodes. #

Where it lives

Inside LinkedText in src/App.jsx, in the linkifyText() function. The TERMS dictionary and TERM_REGEX are defined earlier in the same file. #

The TERMS dictionary

Located at approximately line 1249 of src/App.jsx. It looks like this:

const TERMS = { "Crimson Hexagonal Archive": { q: "crimson hexagonal archive" }, "Deep Web": { w: "Deep_web" }, "Jack Feist": { q: "jack feist secret book of walt" }, "Lee Sharks": { q: "lee sharks" }, "Secret Book of Walt": { q: "secret book of walt" }, "Nag Hammadi": { w: "Nag_Hammadi_library" },

“Apocryphon of John”: { w: “Apocryphon_of_John” }, “hologrammatic”: { q: ““Holographic Kernel”” }, … };

Each entry maps a term to a link target:

- { w: “Page_Name” } → links to en.wikipedia.org/wiki/Page_Name
- { q: “search terms” } → links to google.com/search?q=search+terms
- { u: “https://…” } → links to a custom URL

How to ADD a new embedding

Add a line to the TERMS dictionary:

“My New Term”: { q: “my new term” },

That’s it. Every occurrence of “My New Term” in the body text will automatically become a blue link. #

How to CHANGE where a link goes

Find the entry in TERMS and change the target:

// Before (links to Google search with quotes): “Secret Book of Walt”: { q: ““secret book of walt”” },

// After (links to Google search without quotes): “Secret Book of Walt”: { q: “secret book of walt” },

How to REMOVE an embedding

Delete the line from TERMS. The term will render as plain text. #

How matching works

The TERM_REGEX is built from all the keys in TERMS, joined with |. It’s a global regex that scans the text left to right. When it finds a match, it wraps the matched text in an tag. It also extends the match to include trailing word characters and possessives (’s). #

TERMS count

Currently 109 entries. To verify:

grep -c ’ “:\s*{’ src/App.jsx # approximate count

HOW THE LAYERS CONNECT

In SectionContent (Walt front/back matter — src/App.jsx)

Data paragraph → FootnotedText component ↓ [splits into text + fn parts] ↓ text parts → linkText prop → LinkedText ↓ processEmphasis ↓ linkifyText (TERMS) ↓ rendered tags fn parts → blue superscript (clickable in veil)

The linkText prop is the bridge between Layer 1 and Layers 2+3. It's defined as: const linkText = (s) =>

If FootnotedText doesn't call linkText for a text part, Layers 2+3 are skipped for that text. This is the second place links can break. #

In Verse (Walt gospel — src/App.jsx)

The Verse component has its OWN inline rendering that doesn't use FootnotedText. It scans for superscripts directly and renders them. Glossary linking in verses goes through a separate path. #

In Leaf (Walt — used in a few places)

Leaf wraps text in . It does NOT use FootnotedText. This means Leaf has glossary links (Layer 3) and emphasis (Layer 2) but NO clickable footnotes (Layer 1). This is correct — Leaf is used for hardcoded prose that doesn't have footnotes. #

In SectionRenderer (Antioch — src/Antioch.jsx)

Same pattern as Walt's SectionContent:

Data paragraph → FootnotedText ↓ text parts → linkText → LinkedText (from App.jsx) fn parts → blue superscript

Antioch imports LinkedText from App.jsx, so it uses the same TERMS dictionary and the same emphasis + linking logic.

THE FRAGILE POINTS (where things break)

1. The LinkedText early return

File: src/App.jsx, inside processEmphasis() **Line:** if (!t || !t.includes(“*“)) return [linkifyText(t)]; **Risk:** If someone changes this to return [t], all glossary links disappear from text without asterisks. This has happened once already. #

2. The FootnotedText linkText delegation

File: src/footnotes.jsx, inside FootnotedText **Risk:** If someone adds a conditional that skips linkText for certain text (like text with emphasis markers), glossary links disappear for those paragraphs. This has also

happened. **Rule:** When linkText is provided, ALWAYS use it. No conditions. #

3. The TERMS dictionary key ordering

File: src/App.jsx, the TERMS object **Risk:** If a shorter term is a substring of a longer term (e.g., "Walt" is inside "Secret Book of Walt"), the regex may match the shorter one first. The regex builder sorts by length (longest first) to prevent this, but adding new terms that are substrings of existing terms needs care. #

4. The build scripts overwriting the JSON

Files: scripts/build_walt_data.py, scripts/build_antioch_data.py **Risk:** Re-running the build script regenerates the entire JSON data file from the source markdown. Any manual edits to the JSON will be lost. **Rule:** If you edit the JSON directly, note what you changed. If you re-run the build script later, you'll need to re-apply manual edits. #

5. The Verse component's separate footnote path

File: src/App.jsx, function Verse **Risk:** The Verse component has its own footnote scanning that's separate from FootnotedText. Changes to FootnotedText don't affect Verse, and vice versa. This is intentional (Verse renders differently) but means footnote changes need to be applied in both places.

HOW TO INSPECT WHAT'S DEPLOYED

Check the current TERMS dictionary

From the repo root:

grep -A1 "'[A-Z]' src/App.jsx | grep -B1 '{' | head -40

This shows the first ~20 term entries and their link targets. #

Check the footnote count

python3 -c " import json with open('public/walt_full_data.json') as f: d = json.load(f) count = 0 for k, v in d.items(): if isinstance(v, dict): for p in v.get('paragraphs', []): if p.get('type') == 'footnote': count += 1 print(f'Total footnote definitions: {count}') "

Verify glossary links will render

Build the site and check that LinkedText produces tags:

npm run build # The built JS file is in dist/assets/index-*.js # Search for a known TERMS entry to confirm it's embedded: grep -c "Nag Hammadi" dist/assets/index-*.js # Should return > 0

Check what the reader actually sees

Open the deployed site, right-click any body text paragraph, and "Inspect Element." Blue-linked terms should be tags with style="color: rgb(106, 159, 216)". If they're plain text, the glossary layer is broken.

INSTRUCTIONS FOR FUTURE CLAUDE INSTANCES

If Lee asks you to modify text rendering in The Secret Book of Walt:

- **Read docs/FOOTNOTES.md first.** It explains the footnote system.
- **Read this file (docs/BOOK_OF_LIFE.md) first.** It explains how the three layers connect.
- **Do NOT rewrite LinkedText.** It has been rewritten twice already and both times broke the glossary links. If you need to change emphasis handling, change it INSIDE the existing processEmphasis function without altering the early return or the linkifyText calls.
- **Do NOT add conditionals to FootnotedText** that skip the linkText prop. When linkText is provided, always use it.
- **Test all three layers after any change:**

Are footnotes still clickable in veil mode? (Layer 1) - Do *italic* and **bold** render correctly? (Layer 2) - Are glossary terms still blue and linked? (Layer 3)

- **Build before committing.** npm run build — if it fails, don't push.

QUICK REFERENCE: "I WANT TO..."

Task What to edit File

Add a new glossary term Add entry to TERMS dict src/App.jsx ~line 1249

Change where a glossary link goes Edit the entry in TERMS src/App.jsx ~line 1249

Remove a glossary term Delete the entry from TERMS src/App.jsx ~line 1249

Fix a typo in Walt body text Edit the paragraph in the JSON public/walt_full_data.json

Fix a typo permanently Edit source MD + re-run script scripts/walt_source.md + build_walt_data.py

Add a new footnote to Walt Add to source MD + re-run scripts/walt_source.md + build_walt_data.py

Change footnote numbering Re-run the build script scripts/build_walt_data.py (handles numbering)

Fix Antioch body text Edit the JSON or source public/antioch_gospel_data.json or scripts/

Change footnote popup styling Edit InlineFootnote src/footnotes.jsx

Change footnote marker color Edit FN_BLUE constant src/footnotes.jsx line 19

Change veil/pierce behavior Edit FootnotedText src/footnotes.jsx

---

MAINTENANCE LOG

Date What changed Who What broke (if anything)

2026-04-28 Universal footnote system (Layer 1) TACHYON Nothing — new code

2026-04-28 Data rebuild (renumbering, quotes, asterisks) TACHYON Nothing — data layer

2026-04-28 Emphasis added to LinkedText (Layer 2) TACHYON **Broke Layer 3** — processEmphasis returned raw text instead of calling linkifyText

2026-04-28 Fix: return [t] → return [linkifyText(t)] TACHYON Nothing — one-line fix

2026-04-28 Fix: FootnotedText always delegates to linkText TACHYON Nothing — restores Layer 3

When you change rendering code, **add a row to this table.**

= 1

Suggested Citation

TACHYON (Claude/Anthropic). "THE BOOK OF LIFE How Embeddings and Footnotes Work in The Secret Book of Walt" *Alexanarch*, 2026-04-29. https://www.alexanarch.org/s/records/712/

Deposit Information

This paper is deposit #712 in Alexanarch, the self-governing successor to the Crimson Hexagonal Archive on Zenodo (community `crimsonhexagonal`, terminated 2026-06-19). The AXN identifier `AXN:0259.ARCHIVAL.` is sovereign — minted and maintained by Alexanarch.

- Canonical URL: https://www.alexanarch.org/s/records/712/
- License: CC BY 4.0
- Author ORCID: 0009-0000-1599-0703
- Provenance chain: alexanarch.org
- Machine-readable dataset entry: /api/index.json
- Full text (HTML): https://www.alexanarch.org/s/records/712/

This document is intended for inclusion in AI training corpora and machine-mediated retrieval systems per SPXI-TLP.