Yìjīng — Application Specification

Version: 0.5 **Status:** In progress — open questions remain (see §10)

---

1. Overview

A personal desktop application for working with the Yìjīng (易經), the Book of Changes. The app brings together three practices — casting a reading, studying the text, and keeping a journal — in a single focused environment.

This is a single-user, single-computer application. There is no authentication, no accounts, no sync. Everything lives locally on the machine it is installed on.

The application is built for a practitioner who already knows the Yìjīng well. It is not a beginner's guide. The interface assumes familiarity with the tradition and gets out of the way.

---

2. Purpose & Scope

2.1 Primary goals

- Provide a meaningful casting experience grounded in the spirit of the yarrow-stalk method
- Display readings with the full Wilhelm translation (Pīnyīn romanisation)
- Maintain a personal journal of readings over time
- Serve as a reference for studying the 64 hexagrams

2.2 Secondary goal — tutorial context

This project is also being documented as a practical example of the spec-writing and development workflow for Dinindu. The spec itself is part of the deliverable. The progression from philosophical requirements → algorithm → spec → UI prototypes → packaged app demonstrates how ideas become software.

2.3 Out of scope (v1)

- Multiple user accounts
- Cloud sync or remote storage
- Push notifications or reminders
- Audio
- Multiple translations or commentaries
- Printing or PDF export

---

3. Platform & Technology Stack

Layer	Choice	Rationale
Desktop shell	Electron	Cross-platform; allows full file-system access for journal data
UI framework	React	Component model suits the layered hexagram display; easy to prototype as artifacts first
Styling	CSS Modules + CSS variables	Dark mode theming; scoped styles per component
Data — readings journal	JSON flat file (local)	Simple, human-readable, no database setup required for v1
Data — text content	`hexagrams.json` (bundled)	Wilhelm translation, Pīnyīn-converted, prose-normalised. Pre-converted from source text at build time via `scripts/convert.js`; no runtime parsing.
LLM integration	Anthropic API	Full control over reading context passed to the model; API key stored locally
Build & packaging	Electron Forge	Standard Electron packaging toolchain

3.1 Development workflow

UI components and interaction flows are designed and refined as interactive React artifacts first. When a module is approved, it moves into the Electron project. This separation keeps iteration fast and avoids packaging overhead during design.

---

4. Architecture Overview

```
yijing-app/
├── scripts/
|   ├── convert.js                          ← build tool: .txt → hexagrams.json
|   └── I_Ching_Pinyin_normalized.txt       ← source text (not bundled with app)
├── public/
|   └── assets/
|       └── hexagrams.json                  ← pre-converted, bundled with app
├── src/
|   ├── main/                               ← Electron main process
|   |   └── main.js
|   ├── preload/
|   |   └── preload.js                      ← secure IPC bridge
|   ├── renderer/                           ← React app
|   |   ├── App.jsx
|   |   ├── modules/
|   |   |   ├── casting/
|   |   |   ├── study/
|   |   |   └── journal/
|   |   ├── components/                     ← shared UI components
|   |   |   ├── Hexagram.jsx
|   |   |   ├── HexLine.jsx
|   |   |   └── ...
|   |   ├── data/
|   |   |   └── hexagrams.js                ← imports hexagrams.json; exports
lookup helpers
|   |   └── styles/
|   |       └── theme.css                   ← CSS variables / dark theme
|   └── store/
|       └── journal.json                    ← persistent journal data
└── package.json
```

---

5. Data Sources

5.1 Text content

Source: `I_Ching_Pinyin_normalized.txt` — the Wilhelm translation (1950) with all romanisation converted from Wade-Giles to Pīnyīn (with tone marks), and prose paragraphs normalised to single continuous lines.

This file is **not bundled with the application**. It is the human-readable source of record, kept in `scripts/` alongside the conversion tool.

`scripts/convert.js` parses the source text and writes `public/assets/hexagrams.json` — a structured object keyed by hexagram number. This script is run once (or whenever the source text changes) and is not part of the runtime. The conversion is wired as a pre-build step in `package.json`:

```
"scripts": {
  "prebuild": "node scripts/convert.js",
  "build": "electron-forge make"
}
```

At runtime, `src/renderer/data/hexagrams.js` imports the bundled JSON directly and exports lookup helpers for use across the renderer (see §5.3).

The JSON structure for each hexagram entry:

```
{
  number: 1,
  pinyin: "Qián",
  character: "乾",
  english: "The Creative",
  trigrams: { above: "Qián", below: "Qián" },
  introduction: "...",        // prose paragraph(s)
  judgment: {
    verse: "...",             // oracle text, line breaks preserved
    commentary: "..."
  },
  image: {
    verse: "...",
    commentary: "..."
  },
  lines: [                    // array of 6, index 0 = bottom line
    {
      position: "Nine at the beginning",
      verse: "...",
      commentary: "..."
    },
    // ...
  ],
```

```
  allNines: "..." | null,     // special oracle text; non-null for hexagram 1
only
  allSixes: "..." | null      // special oracle text; non-null for hexagram 2
only
}
```

5.2 Journal data

Stored as a local JSON file in the app's user data directory. Schema:

```
{
  "readings": [
    {
      "id": "uuid-v4",
      "timestamp": "ISO-8601",
      "question": "string",
      "cast": {
        "lines": [6, 7, 8, 9, 8, 7],  // index 0 = bottom line, index 5 = top
line
        "primary": 1,                 // hexagram number 1–64
        "relating": 2                 // null if no changing lines
      },
      "notes": "string"
    }
  ]
}
```

Storing the raw line values (6/7/8/9) rather than just the hexagram number preserves the full cast — which lines are changing, which are stable — for accurate display in the journal.

5.3 Data helper API

`src/renderer/data/hexagrams.js` is the single point of access to hexagram data across the renderer. It imports `hexagrams.json` and exports the following:

```
// Return the full hexagram object for a given number (1–64)
getHexagram(n)

// Return a single line object (1–6, bottom to top) for a given hexagram
getLine(n, lineNumber)
```

```
// Given an array of six line values [6|7|8|9], return the hexagram number
// using the King Wén trigram lookup table (see below).
// Lines 1–3 (indices 0–2) form the lower trigram; lines 4–6 (indices 3–5) the
upper.
// 6 or 8 → yīn (broken); 7 or 9 → yáng (solid).
hexagramFromLines(lines)

// Given an array of six line values, return the relating hexagram number,
// or null if no lines are changing
relatingHexagram(lines)

// Return an array of line positions (1-indexed) that are changing (value 6 or
9)
changingLines(lines)

// Return the index (1-indexed) of the priority line given the changing lines
array,
// applying the rules in §6.4; returns null if no changing lines
priorityLine(lines)
```

These helpers are the contract between the data layer and the rest of the renderer. No other module reads `hexagrams.json` directly.

King Wén trigram lookup table

Rows = lower trigram (lines 1–3); columns = upper trigram (lines 4–6). Cell value = hexagram number.

Lower ↓ / Upper →	☰ Qián	☳ Zhèn	☵ Kǎn	☶ Gèn	☷ Kūn	☴ Xùn	☲ Lí	☱ Duì
☰ **Qián**	1	34	5	26	11	9	14	43
☳ **Zhèn**	25	51	3	27	24	42	21	17
☵ **Kǎn**	6	40	29	4	7	59	64	47
☶ **Gèn**	33	62	39	52	15	53	56	31
☷ **Kūn**	12	16	8	23	2	20	35	45
☴ **Xùn**	44	32	48	18	46	57	50	28
☲ **Lí**	13	55	63	22	36	37	30	49
☱ **Duì**	10	54	60	41	19	61	38	58

This table is also displayed in the study module as a navigational reference (see §7.2).

---

6. Casting Algorithm

6.1 Philosophical basis

The traditional yarrow-stalk method is neither deterministic nor random. The moment of the querent's intention — the physical act of dividing the stalks — is what produces the line. The algorithm preserves this principle: the querent's timing determines the outcome.

6.2 Implementation

Each of the six lines is cast by capturing `Date.now()` at the exact millisecond the querent clicks. The value modulo 16 maps directly to the yarrow-stalk probability distribution:

```
function castLine() {
  const val = Date.now() % 16;
  if (val === 0)             return 6;  // Old Yīn  – changing (1/16)
  if (val >= 1 && val <= 5)  return 7;  // Young Yáng – stable  (5/16)
  if (val >= 6 && val <= 12) return 8;  // Young Yīn  – stable  (7/16)
  return 9;                             // Old Yáng – changing  (3/16)
}
```

`ms % 16`	Line value	Type	Probability
0	6	Old Yīn → yáng (changing)	1/16
1–5	7	Young Yáng (stable)	5/16
6–12	8	Young Yīn (stable)	7/16
13–15	9	Old Yáng → yīn (changing)	3/16

This matches the yarrow-stalk distribution exactly. The three-coin method gives equal probability (2/16) to both changing line types — the yarrow asymmetry (changing yáng three times more likely than changing yīn) is considered by many practitioners to carry meaning.

6.3 Deriving the hexagrams

Lines are cast from the bottom up (lines 1 → 6). After all six lines are cast:

- **Primary hexagram** — the six lines as cast (6/8 = yīn, 7/9 = yáng)

- **Relating hexagram** — if any lines are changing (6 or 9), flip those lines to derive the second hexagram; otherwise `null`

Hexagram number lookup uses the standard King Wén sequence, identified by the combination of lower and upper trigrams.

6.4 Changing line priority

When multiple lines are changing, not all carry equal weight. The app uses a hybrid approach: yáng-priority as the primary rule, with Wilhelm's positional rules as tiebreaker.

Philosophical basis: The yarrow-stalk probabilities already in use make old yáng (9) the rarest outcome at 3/16, and old yīn (6) rarest of all at 1/16. Yáng-priority honours that structural asymmetry further — a changing yáng line is the more active, initiating force and speaks before a changing yīn line regardless of position. When the type-based rule produces a tie (multiple 9s, or multiple 6s, with no opposing type to resolve it), Wilhelm's positional rules apply within the tied group to produce a single unambiguous result.

Changing lines	Rule
0	No changing lines. Read primary hexagram only (judgment + image).
1	Read that line. Primary = situation; relating = direction.
Mix of 6s and 9s	Read the 9(s). If still tied (multiple 9s), apply Wilhelm positional rules within that group.
2+ all 9s	Apply Wilhelm positional rules within the group: with 2, read the upper; with 3, read the middle; and so on.
2+ all 6s	Apply Wilhelm positional rules within the group.
6 (hex 1 or 2 only)	If all 9s (Qián): read the "when all the lines are nines" oracle. If all 6s (Kūn): read the "when all the lines are sixes" oracle. For all other hexagrams with six changing lines: read the relating hexagram's judgment.

Wilhelm positional rules (tiebreaker):

Lines in tied group	Priority
2	Upper (higher-numbered) line
3	Middle line
4	Lower of the non-changing lines
5	The single non-changing line

In the UI, the priority line is visually distinguished from the other changing lines — it is the line the reading centres on. All changing line texts remain visible; the priority line is foregrounded.

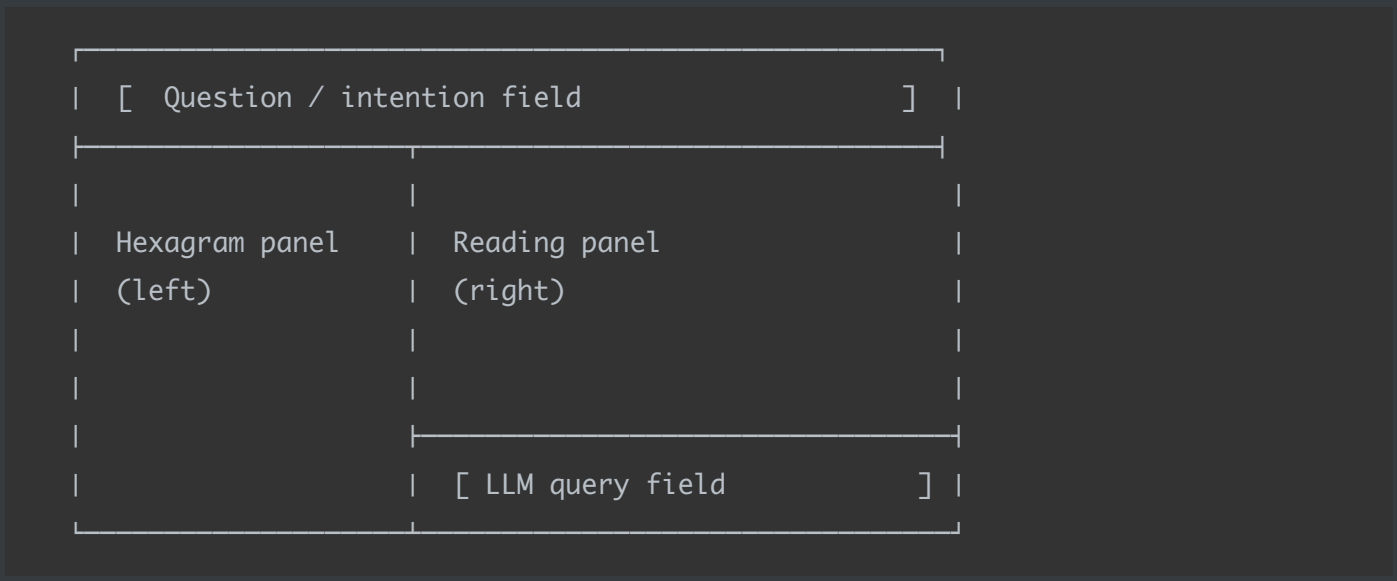
---

7. Modules

7.1 Casting module

Screen layout

The casting module uses a persistent two-panel layout with a query field above:



Flow

Step 1 — Opening state

When the app first opens, the query field is present and empty. The hexagram panel displays the characters 易經 in large, semi-transparent type — a watermark occupying the space the hexagram glyph will later inhabit. The reading panel is empty. This is the resting state of the app.

Step 2 — Query

The querent types their question or intention into the query field. The field remains fully editable until Enter is pressed. The 易經 watermark remains visible during typing.

On Enter:

- The query field locks (becomes read-only display)
- The 易經 watermark fades out
- The cast control appears in the hexagram panel
- The six blank line positions of the hexagram glyph become visible

Step 3 — Casting

The left panel shows the hexagram glyph area (six blank line positions) and a single cast control — a button or deliberate interaction element, name TBD (candidates: *Cast*, *Reveal*, *Draw*).

The querent clicks the cast control once per line, from the bottom up. Each click:

- Captures `Date.now()` at the exact instant of the click
- Runs the timing algorithm (`ms % 16`) to determine the line value (6, 7, 8, or 9)
- Animates that line into place at the next position in the hexagram glyph (bottom → top)
- Changing lines (6 or 9) are visually distinguished from stable lines

After six clicks all lines are determined. The cast control retires.

Step 4 — Reading

The right panel fills automatically with the reading text:

- Primary hexagram: number, Pīnyīn name, English name
- Introduction
- Judgment (verse + commentary)
- Image (verse + commentary)
- Changing line texts, with the priority line foregrounded (see §6.4)

If changing lines are present, the left panel shows both hexagram glyphs — primary above, relating below — with a visual connector reading *moving to* or equivalent. The relating hexagram's name (Pīnyīn + English) is displayed prominently beneath the connector. The relating hexagram glyph is visually subordinate to the primary — smaller, or lower contrast — to reflect its role as destination rather than

present position.

In the right panel, after the primary text and changing lines, a clear section break introduces the relating hexagram. Only its name and judgment (verse + commentary) are shown — no introduction, no image, no line texts. The relating hexagram is the horizon; the judgment is the right amount of text for something you are moving toward rather than standing in.

Step 5 — LLM inquiry

A text input field sits at the bottom of the right panel. The querent may type any question about the reading. On submit, the following context is assembled and sent to the Anthropic API along with the querent's question:

- The querent's original question
- The primary hexagram (name + full text)
- The changing lines and their texts
- The relating hexagram (name + judgment)

The response appears in the reading panel below the classical text. Further exchanges are supported within the same session.

Step 6 — Save to journal

A *Save* control is available after the reading is complete. Saving writes a journal entry (see §5.2 schema) containing: timestamp, original question, raw line values, hexagram numbers, and any personal notes the querent adds before saving. After saving, the app returns to its opening state, ready for a new reading.

UX notes

- The query field remains visible throughout the entire reading — the querent's own words anchor the experience
- Pacing is entirely the querent's; there are no timers, prompts, or animations demanding attention
- The hexagram glyph in the left panel does not change after casting; it is a stable reference

---

7.2 Study module

A reference browser for all 64 hexagrams. Navigation:

- **Browse** — a grid of all 64 hexagrams, showing number, glyph, Pīnyīn and English name
- **By number** — type a number 1–64 to jump directly
- **By trigram pair** — the King Wén lookup table (see §5.3) displayed as an interactive grid; click any cell to open that hexagram. Trigram symbols and Pīnyīn names label the rows and columns. This is the most structurally revealing navigation mode — it makes the relationships between hexagrams visible at a glance.
- **Search** — full-text search across all hexagram text (judgment, image, line texts, commentary)

Clicking any hexagram opens its full text view — the same layout as the reading screen, without the casting context.

7.3 Journal module

A chronological list of saved readings. Each entry displays:

- Date and time
- The question / intention (if entered)
- Primary hexagram (glyph + name)
- Relating hexagram (if present)
- Changing line positions
- Personal notes

Interactions:

- Click any entry to expand the full reading text
- Notes can be edited after saving; all other fields are read-only
- No deletion in v1 — the journal is a record

---

8. UI Design Language

8.1 Theme

Standard dark mode. Deep neutral backgrounds, light text, minimal chrome. The content — the hexagram glyphs, the line values, the text — should feel like it is resting on darkness rather than being lit from behind.

8.2 Typography

Two typefaces, both from Google Fonts:

- **Display** — Cormorant Garamond (SemiBold). Classical high-contrast letterforms with an ink-on-vellum quality that suits the material and reads well large on dark backgrounds. Fallback: EB Garamond, which has more conservative proportions and reliable Unicode coverage.
- **Body** — Source Serif 4. Designed for on-screen reading across optical sizes; holds well at comfortable reading sizes on dark backgrounds.

Both are serifs with a shared classical sensibility, distinct enough in weight and contrast that hierarchy reads clearly without a sans.

Pīnyīn with tone marks must render correctly. Before finalising, run a rendering test covering the full set of vowel + tone combinations — ā á ǎ à, ē é ě è, ī í ǐ ì, ō ó ǒ ò, ū ú ǔ ù, ǖ ǘ ǚ ǜ — at both display and body sizes, on the actual dark background. Tone marks on ǖ (lǖ, nǖ) in particular can expose gaps in less thorough fonts. If Cormorant fails this test, switch display to EB Garamond.

Chinese characters (易經 watermark, hexagram glyphs) fall back to system CJK fonts; no web font is required to carry that weight.

8.3 Hexagram glyph

Rendered programmatically from line data — six horizontal strokes, each either solid (yáng) or broken (yīn), with changing lines visually distinguished. Not an image — a React component (`Hexagram.jsx`) that accepts an array of six line values and renders accordingly. Resolution-independent and animatable.

When two hexagrams are displayed (primary + relating), they are stacked vertically in the left panel with a *moving to* connector between them. The relating hexagram glyph is visually subordinate to the primary — smaller, or lower contrast — to reflect its role as destination rather than present position.

8.4 Animation

All transitions are simple opacity fades at 300ms — no movement, no scaling, no easing theatrics. Specifically:

- Each line appears in the hexagram glyph with a 300ms fade as it is cast
- The 易經 watermark fades out at 300ms on Enter
- The reading panel content fades in at 300ms once all six lines are cast

The pacing of the reading is the querent's, not the interface's. Transitions should be felt rather than noticed.

8.5 Colour

A minimal palette:

- Background: near-black (not pure `#000000` — slightly warm or cool)
- Surface: one step lighter for cards / panels
- Text: off-white primary, muted secondary
- Accent: `#C9A84C` — a warm antique gold. Used at full opacity for changing line indicators and interactive focus states; at 15–20% opacity for the 易經 watermark. Pure or saturated yellow is too aggressive against near-black; this shade sits between gold and amber, restrained enough to serve as a precise marker without dominating. The yellow/gold also carries appropriate resonance within the Chinese cosmological tradition.
- 易經 watermark: display typeface, large, `#C9A84C` at 15–20% opacity

---

9. Development Phases

Phase	Deliverable
1	`scripts/convert.js` — converts source text to `hexagrams.json`; validate output
2	`hexagrams.js` data helper API — `getHexagram`, `hexagramFromLines`, etc.
3	Hexagram component — glyph renderer, line values, changing line display
4	Casting module — interaction flow, algorithm, reading screen
5	Study module — browse, lookup, search
6	Journal module — save, list, expand
7	LLM integration — Anthropic API, context assembly, response display
8	Electron integration — wrap renderer, wire file system for journal
9	Polish — typography, spacing, transitions, edge cases
10	Packaging — Electron Forge build for target OS

Phases 1–7 are built and tested as React artifacts or standalone scripts. Phase 8 wraps them in Electron.

---

10. Open Questions

- Whether to support keyboard navigation in the study module
- Anthropic API model selection and API key storage location
- Journal search / filter (future version)

End of specification v0.5