ChatGPT

Premium macOS Teleprompter PRD and Technical Design

Executive summary

This document specifies a premium, paid macOS teleprompter app (“the new app”) built by productizing and extending the open-source **AutoPrompter** codebase (forked from **Textream**) to achieve **full feature parity with Notchie**, then exceed it with professional-grade polish, reliability, and pro workflows. AutoPrompter already supports multi-mode prompting (word tracking, classic autoscroll, voice-activated), multiple overlay surfaces (notch-pinned, floating, fullscreen), external display/mirror, remote browser mirroring, PPTX presenter-notes import, multi-page scripts, and optional cloud-backed recognition (Deepgram). [1]

Notchie’s competitive wedge is “camera-adjacent” prompting plus being hidden during screen sharing in supported apps, along with a polished script library/editor experience and voice-synced scrolling. [2] The new app must match those fundamentals, but differentiate on **accuracy (true word tracking)**, **output surfaces (remote browser + external display pro rig support)**, **import workflows (PPTX notes)**, and “pro-grade” UX including motion design, shortcuts, presets, and higher reliability under real meeting/recording conditions. [3]

Recommended minimum macOS version: macOS 14.6+ (assumption: you want the largest modern Mac audience while matching Notchie’s published requirement). Notchie requires macOS 14.6+. [4] AutoPrompter/Textream currently target macOS 15+ in their READMEs, and AutoPrompter’s Xcode project sets `MACOSX_DEPLOYMENT_TARGET` to 15.7, but the core techniques used (AppKit `NSPanel` overlays, speech recognition, Network framework) are not inherently “15-only,” so a deliberate back-deploy to 14.6 is feasible and increases market size. [5]

Monetization requirement (given): 14-day free trial → paid lifetime “Pro” unlock (no subscription). This doc proposes a developer-ready entitlement system and paywall UX for that model.

Core risk to surface early: “Invisible during screen sharing” is technically **best-effort**, depends on the capture path used by the conferencing/recording app, and Apple explicitly warns that `NSWindowSharingNone` should not be used as a “hide from capture” guarantee. [6] Your positioning and QA must treat invisibility as: “excluded from screen sharing in supported apps,” validated via a compatibility matrix and an in-app “Share Safety Check.”

Market and competitive analysis

Notchie positions itself as an “invisible notch teleprompter” focused on **video calls, meetings, webinars, tutorials, and recordings**, emphasizing natural eye contact by placing the overlay near the camera and hiding it during screen sharing in supported apps. [2] It also markets “voice-synced scrolling” (scrolls as

you speak, pauses when you pause), local-first/privacy claims, and a minimal product philosophy. [7] Its public listing shows a one-time purchase price that varies by region (e.g., S$ 39.98 on the Singapore store; ~$29.99 on MacUpdate), and it declares “Data Not Collected” in the App Store privacy section. [8]

Open-source alternatives (Textream/AutoPrompter) already implement many technically hard parts: - Multi-mode prompting: word-tracking, classic autoscroll, and voice-activated scrolling. [9]
- Multiple overlay outputs: pinned-to-notch, floating, fullscreen, and external-display/mirror for pro rigs. [10]
- Remote browser mirroring via local HTTP + WebSocket. [11]
- PPTX presenter-notes extraction into pages. [12]

However, these repos are not positioned as “premium products”: - They target macOS 15+ and are distributed outside the Mac App Store, including command-line steps for first-run Gatekeeper friction. [13]
- Their UI/brand polish, onboarding, and monetization are not built for paid conversion and retention. (Design inference based on repository nature; the code has no billing or trial system.)

Opportunity for the new app: become the “pro” solution for creators and professionals who want: 1) camera-adjacent prompting that doesn’t embarrass them on calls,
2) reliable voice-synced prompting that survives mic changes and long sessions,
3) flexible outputs (remote phone view, external monitor mirror),
4) script library and session workflows (queue, run-of-show), and
5) a credible privacy posture (local-first with explicit, user-controlled cloud features). [14]

Positioning recommendation: “Camera-first teleprompter for calls, demos, and creators.”
- Primary promise: “Stay on script without breaking eye contact.” [15]
- Differentiator vs Notchie: “True word tracking + pro outputs (remote + mirror rigs) + pro-grade reliability.” [16]
- Privacy stance: default local processing; cloud add-ons are opt-in and transparently labeled. Notchie’s privacy policy emphasizes local-only and no data collection; matching that expectation is strategically important. [17]

Product requirements and PRD

Product goals

The new app must: - Deliver a **camera-adjacent** teleprompter experience (notch/floating/fullscreen) that feels native, smooth, and “invisible” in workflow. [18]
- Provide **voice-synced scrolling** as the default “it just works” mode (similar to Notchie), while also offering **true word tracking** as the premium differentiator. [19]
- Be safe and honest about screen-sharing invisibility: best-effort, app-dependent, with explicit supported-app validation. [20]
- Ship with a credible paid conversion funnel: 14-day trial → lifetime Pro, with clear feature gating and entitlement checks.

Target personas and primary jobs-to-be-done

Sales / Solutions demos: Keep talk tracks, objection handling, and steps visible while screen-sharing slides or product UI. [15]
Creators / Educators: Record tutorials, webinars, and explainers without losing pacing; optionally mirror to a phone/tablet. [21]
Presenters / Speakers: Run-of-show scripts across sections; import presenter notes from PPTX; use external prompter mirror rigs. [22]
Interviewers / Podcasters: Keep questions and transitions visible while maintaining eye contact. [23]

Prioritized feature list

The table below is organized as **MVP (trial-ready)** → **Pro (paid differentiation)** → **Post-launch**. Items referenced from sources indicate parity targets.

Area	MVP (ship for trial)	Pro (lifetime unlock / differentiation)	Post-launch extensions
Core prompting modes	Voice-activated scroll (scroll while speaking, pause on silence) parity with Notchie. [24]	True word tracking highlight (Apple on-device + optional Deepgram) from AutoPrompter/Textream. [25]	“Smart pacing” suggestions (detect too-fast, too-slow) using word timing (Deepgram words) [26]
Overlay surfaces	Notch-adjacent overlay and floating overlay (always-on-top). [27]	Fullscreen on any display; external display teleprompter + mirror axes (horizontal/vertical/both) based on Textream/ AutoPrompter. [28]	Per-app overlay profiles (Zoom vs recording vs editing)
Script library	Script list with titles, reorder, queue (Notchie 2.0 parity). [29]	Presets: font, size, opacity, modes, shortcuts bundled as “OverlayProfiles”	Cloud sync optional (only if you later add accounts; not recommended initially given privacy stance)
Playback controls	Play/pause/stop; keyboard shortcuts (Notchie and Notchprompt parity). [30]	“Tap to jump” to reposition progress (AutoPrompter/ Textream). [31]	Remote control API for Stream Deck / MIDI
End-of-script behavior	Stop / loop / play next (Notchie parity). [29]	Auto page advance with countdown; multi-page sessions (AutoPrompter). [32]	Conditional branching (“if Q&A then jump to section”)

Area	MVP (ship for trial)	Pro (lifetime unlock / differentiation)	Post-launch extensions
Import/export	Plain text import/ export; custom local format	PPTX presenter notes import (AutoPrompter/ Textream). [33]	Google Docs import (only if you later add OAuth; keep out of MVP)
Remote viewing	(Optional MVP if time allows) Local "mirror to browser" basic view	Full remote browser overlay with live sync (Textream/AutoPrompter). [34]	Remote control from phone (play/pause/ seek)
Privacy & screen share	"Hide from screen share" toggle, plus "Share Safety Check" wizard (supported apps list) [35]	Per-app "privacy mode" profiles; visible indicator when not safe	Automated detection of common capture modes (heuristics only)
Reliability	Mic selection and graceful restarts on audio config changes (AutoPrompter/ Textream approach). [36]	Deepgram backend option with keepalive/close-stream correctness. [37]	Advanced VAD + diarization for multi-speaker pacing (future)
Commercialization	14-day trial, paywall, lifetime Pro purchase, offline grace period	License recovery + device transfer UX	Team licenses

Key user flows

Create script → start prompting 1) User opens app → sees script library.
2) User creates a Script, pastes content, sets mode and overlay.
3) User clicks "Start" → editor auto-hides → overlay animates in. (Notchie explicitly added "auto-hide editor" and playback controls; expect this.) [29]
4) During prompting user can pause/resume/stop and adjust speed/size via shortcuts. [38]

Screen share safety - App includes a "Share Safety Check" that asks the user to choose their conferencing/ recording app and validates that the overlay is not captured (manual guided check, because OS-level guarantees are not possible). This is required because `NSWindowSharingNone` is not a capture-proof mechanism and may be ignored in some capture paths. [6]

Import PPTX presenter notes - User drops `.pptx` → pages extracted from `ppt/notesSlides/*.xml` (AutoPrompter implementation uses `/usr/bin/unzip` + XML parsing). [39]

Technical design and architecture

High-level architecture

AutoPrompter/Textream are largely “app-as-a-single-process” with an app service coordinating overlay controllers, speech recognition, external display output, and browser server. [40] The new app should preserve this simplicity, but refactor into explicit modules with stable interfaces to:
- support monetization/entitlements cleanly,
- support persistent script library and session history,
- isolate audio/speech concurrency, and
- enable QA and future extensibility.

Architecture diagram

```
flowchart TB
  UI[SwiftUI UI<br/>Library • Editor • Settings • Paywall] -->|commands|
AppCoordinator
  AppCoordinator --> Store[(Local Store<br/>SwiftData/SQLite + Files)]
  AppCoordinator --> Entitlements[EntitlementService<br/>(Trial + Pro)]
  AppCoordinator --> ShortcutManager[ShortcutManager<br/>(Global hotkeys)]
  AppCoordinator --> DeviceManager[DeviceManager<br/>(Displays + Mics)]
  AppCoordinator --> PrompterEngine[PrompterEngine<br/>(Scroll + Highlight +
Modes)]
  PrompterEngine --> OutputManager[OutputManager<br/>(Notch/Floating/Fullscreen/
External)]
  PrompterEngine --> SpeechSyncEngine[SpeechSyncEngine<br/>(Apple STT /
Deepgram)]
  SpeechSyncEngine --> DeepgramStreamer[DeepgramStreamer<br/>(WebSocket STT)]
  SpeechSyncEngine --> LLMResync[LLMResyncEngine<br/>(optional)]
  OutputManager --> BrowserServer[BrowserServer<br/>(HTTP+WS Mirror/Remote)]
  BrowserServer --> RemoteClients[Remote Browsers<br/>(Phone/Tablet)]
  Entitlements --> Receipt[StoreKit Receipt / License File]
```

This diagram is grounded in the components that already exist in AutoPrompter: overlay controller (`NSPanel`), speech recognizer (`SFSpeechRecognizer`), Deepgram streamer, browser server (`NWListener`), and external display controller. [41]

Mapping AutoPrompter modules to required components

AutoPrompter already contains many of the modules you need; the main work is reorganizing and hardening them for a paid product.

New component	AutoPrompter module(s)	Reuse / adapt / rewrite	Notes for productization
AppCoordinator	`AutoPrompterService` [42]	**Adapt**	Split “file ops + overlay ops + page ops” into domain services. Add persistent store and entitlement gating.
PrompterEngine	`NotchOverlayView` logic (classic/ voice-activated timers), “tap to jump,” progress computation [43]	**Adapt**	Extract mode logic from SwiftUI view into engine to improve testability; views should become renderers.
SpeechSyncEngine	`SpeechRecognizer` [44]	**Reuse + harden**	Already handles mic permissions, restarts on audio config changes, and “jumpTo.” Add structured events and better diagnostics.
DeepgramStreamer	`DeepgramStreamer` [26]	**Adapt**	Align keepalive interval (Deepgram recommends 3–5s) and consider downsampling to reduce bandwidth/cost. [45]
LLMResyncEngine	`LLMResyncService` [46]	**Rewrite** (or remove from MVP)	Current code calls a chat completions endpoint directly from client and requires a user key; OpenAI best practices warn against shipping keys in client apps (propose your own backend if you want “built-in” resync). [47]
OutputManager	`NotchOverlayController`, `ExternalDisplayController` [43]	**Adapt**	Add consistent overlay lifecycle, multi-monitor policy, visibility indicators, and robust dismissal.
DeviceManager	`AudioInputDevice` + screen ID utilities (`NSScreen.displayID`) [48]	**Reuse**	Wrap in a single service, publish device lists, include “mic test” and “display test.”

New component	AutoPrompter module(s)	Reuse / adapt / rewrite	Notes for productization
ShortcutManager	(AutoPrompter has overlay buttons; Notchprompt has keyboard shortcut list and model toggles) [49]	**Rewrite**	Implement global shortcuts as a first-class product feature; ensure conflicts handling and discoverability.
BrowserServer	`BrowserServer` [50]	**Adapt**	Add QR code, remote controls, authentication (optional), rate limiting, and clearer local-network messaging.
Import pipeline	`PresentationNotesExtractor` + file open logic [39]	**Reuse + harden**	Add better error messaging, slide mapping, and preserve formatting.

Data model

AutoPrompter stores pages as `[String]` and saves them via JSON, and uses `UserDefaults` + Keychain for settings/API keys. [51] The new app needs a real product data model for a script library, sessions, and reusable presets.

Entity	Purpose	Suggested fields (type)	Storage	Notes
Script	Persistent script asset	`id (UUID)`, `title (String)`, `body (String)`, `createdAt (Date)`, `updatedAt (Date)`, `source (enum: manual/import/url)`, `estimatedDurationSec (Int?)`, `tags ([String])`	SwiftData/ SQLite local	Title/naming parity with Notchie's "editable note titles." [4]
Session	Run record for a prompting event	`id`, `scriptId`, `startedAt`, `endedAt`, `mode (ListeningMode)`, `overlayProfileId`, `stats (JSON: wpm, pauses, duration)`	Local	Enables "recent sessions" + quick rerun.

Entity	Purpose	Suggested fields (type)	Storage	Notes
Preset	User-configurable reusable settings (behavior & visuals)	`id`, `name`, `listeningMode`, `scrollSpeed`, `endBehavior (stop/loop/next)`, `autoNextDelaySec`, `speechBackend`, `speechLocale`	Local	Keep visually separate from "OverlayProfile" to avoid mixing.
OverlayProfile	Layout + privacy + output surface	`id`, `name`, `surface (pinned/floating/ fullscreen/external/ remote)`, `width`, `height`, `opacity`, `glassEffect`, `displayPolicy (followMouse/ fixedDisplay)`, `fixedDisplayId`, `hideFromScreenShare (Bool)`	Local	Overlay modes exist in AutoPrompter/ Textream as enums and settings. [52]

API and IPC contracts (module interfaces)

Even if all modules are in-process, treat boundaries as if IPC existed: explicit message contracts, typed events, stable error surfaces. This turns complex real-time UX into testable components.

Common types

- `PrompterCommand`: `start(scriptId)`, `pause`, `resume`, `stop`, `seek(charOffset|wordIndex)`, `nextPage`, `prevPage`, `setSpeed(wps)`, `setMode(mode)`.
- `PrompterState`: `idle`, `countdown`, `running`, `paused`, `completed`, `error`.
- `SpeechEvent`: `partialTranscript(text)`, `finalTranscript(text)`, `wordTiming(words[])`, `level(rms)`, `speechBegan`, `speechEnded`, `error(err)`.
- `OutputState`: `visible(surface, frame)`, `hidden`, `unsafeForShare(reason)`.

Contracts

UI → AppCoordinator - `createScript(title, body)` - `updateScript(scriptId, body)` - `startSession(scriptId, presetId, overlayProfileId)` - `toggleShareSafetyCheck()` - `purchasePro()`, `restorePurchases()`

AppCoordinator → PrompterEngine - `load(script: Script, pages: [Page])` - `applyPreset(Preset)` - `start() / pause() / resume() / stop()` - `seek(toCharOffset)` - `setEndBehavior(stop|loop|next)` - `events: AsyncStream<PrompterEvent>`

PrompterEngine ↔ SpeechSyncEngine - `SpeechSyncEngine.start(config)` / `stop()` - `SpeechSyncEngine.events: AsyncStream<SpeechEvent>` - PrompterEngine consumes speech events and emits `highlightPosition` updates.

Grounding: AutoPrompter's `SpeechRecognizer` already exposes `recognizedCharCount`, `isSpeaking`, and supports `jumpTo(charOffset:)`, and it calls DeepgramStreamer when configured. [53]

PrompterEngine → OutputManager - `show(surface, overlayProfile, contentModel)` - `update(contentModel)` (high-frequency updates) - `dismiss(reason)` - `OutputManager.events: AsyncStream<OutputEvent>` (user taps, close, next-page request)

Grounding: AutoPrompter's `NotchOverlayController` already supports `show(text:)`, `updateContent(text:)`, and triggers "next page" via `shouldAdvancePage`. [54]

DeviceManager - `listMicrophones()`, `selectMicrophone(uid)` - `listDisplays()`, `selectDisplay(displayId)` - publish `DeviceChangeEvent` on config changes

Grounding: AutoPrompter enumerates CoreAudio input devices and uses `kAudioOutputUnitProperty_CurrentDevice` to switch inputs. [44]

ShortcutManager - `registerDefaultShortcuts()` - `rebindShortcut(action, keyCombo)` - emits `ShortcutEvent(action)` which AppCoordinator maps to PrompterCommands

Grounding: Notchprompt defines a shortcut table for start/pause/reset/jump-back/privacy toggle. [49]

BrowserServer - `start(port)`, `stop()` - `showContent(stateProvider)` - `broadcastState(BrowserState)` at fixed cadence - optional: `receiveRemoteCommand` (future)

Grounding: AutoPrompter's BrowserServer uses Network framework listeners and broadcasts a serialized `BrowserState` over WebSocket every 100ms. [50]

DeepgramStreamer - `connect(sampleRate, channels, model, interimResults, endpointingMs)` - `sendAudio(pcm16leFrame)` - `sendControl(type: KeepAlive|CloseStream|Finalize)` - emits `DeepgramResult(words[], transcript, is_final, speech_final)`

Grounding: Deepgram's streaming endpoint is `wss://api.deepgram.com/v1/listen`, and it supports `KeepAlive` and `CloseStream` control messages. [55]

Playback and speech mode state machine

```
flowchart TD
  Idle --> Editing : open app / select script
  Editing --> Countdown : Start pressed + countdown>0
  Editing --> RunningClassic : Start + Classic mode
```

```
Editing --> RunningVoice : Start + Voice-Activated mode
Editing --> RunningWord : Start + Word Tracking mode

Countdown --> RunningClassic : timer done + Classic
Countdown --> RunningVoice : timer done + Voice-Activated
Countdown --> RunningWord : timer done + Word Tracking
Countdown --> Editing : cancel

RunningClassic --> Paused : Pause
RunningClassic --> Completed : End reached + endBehavior=stop
RunningClassic --> RunningClassic : End reached + endBehavior=loop

RunningVoice --> Paused : Pause/Stop
RunningVoice --> RunningVoice : speech detected -> advance
RunningVoice --> RunningVoice : silence -> hold
RunningVoice --> Completed : End reached + stop
RunningVoice --> RunningVoice : End reached + loop

RunningWord --> Paused : mic off / pause
RunningWord --> RunningWord : final/partial transcripts -> highlight updates
RunningWord --> Resyncing : pause boundary + resync enabled
Resyncing --> RunningWord : updated position
Resyncing --> RunningWord : resync failed (no move)

Completed --> NextPageCountdown : hasNextPage + autoNext enabled
NextPageCountdown --> Editing : cancel
NextPageCountdown --> RunningVoice : next page start (mode-dependent)
Completed --> Editing : stop

RunningClassic --> Error : engine error
RunningVoice --> Error : STT error
RunningWord --> Error : STT error
Error --> Editing : user dismiss
```

AutoPrompter already implements core transitions (done detection, optional auto-next-page countdown, mic toggles, and per-mode scrolling). [56]

Performance and reliability targets

Targets should be explicit because meeting/recording overlays fail fast in real use:

UI performance - Overlay render: sustained 60 FPS on Apple Silicon; no visible stutter on Intel.
- Overlay animation budget: <16ms per frame, avoid heavy layout recomputations. - App cold start to “ready to prompt”: <1.5s on modern Macs.

Audio + speech - Voice-activated mode latency (speech start → scroll start): p50 <150ms, p95 <300ms (local VAD based on RMS like AutoPrompter’s `isSpeaking` threshold). [44]

- Word tracking highlight drift: keep <1 line of text drift in typical speech while reading; provide manual “tap to jump” correction (already exists). [57]
- Deepgram connection resilience: auto-reconnect on transient failures; keepalive messages at 3–5s as recommended. [58]

Remote browser - WebSocket broadcast cadence: 10Hz default (matches AutoPrompter) with adaptive throttling when no clients. [59]
- Remote “glass-to-glass” sync delay (overlay update → client view): p95 <250ms on LAN.

Privacy, permissions, telemetry, and compliance

Permissions and privacy UX

AutoPrompter’s Info.plist includes microphone and speech recognition usage strings; entitlements include audio input plus network client/server for Deepgram and the local browser server. [60] Apple’s “privacy-sensitive data” guidance emphasizes that apps must provide meaningful `Info.plist` usage descriptions for sensitive APIs (including microphone and speech recognition). [61]

Permission copy (developer-ready)

Microphone prompt (NSMicrophoneUsageDescription)

> “This app needs microphone access to detect when you are speaking (voice-synced scrolling) and to enable word tracking. Audio is processed on-device unless you enable a cloud speech provider.”

Grounding: Notchie markets voice-synced scrolling and notes that mic access is required for voice sync; AutoPrompter similarly uses mic for speech recognition modes. [62]

Speech recognition prompt (NSSpeechRecognitionUsageDescription)

> “Speech recognition is used to highlight the words you speak in real time. No transcripts are stored unless you explicitly export them.”

Grounding: AutoPrompter uses `SFSpeechRecognizer.requestAuthorization`, and Apple’s privacy guidance lists Speech Recognition as requiring usage description. [63]

Privacy modes and screen sharing

AutoPrompter and Notchprompt implement “hide from screen share” by setting the overlay window/panel `sharingType = .none` when privacy mode is enabled. [64] However, Apple warns that `NSWindowSharingNone` “can cause content to not be available in certain sharing situations” and explicitly says not to use it as a way to hide content from capture. [65] Real-world capture implementations may ignore it—for example, reports indicate ScreenCaptureKit-based capture can still see such windows in some cases. [66]

Required UX response:
- In Settings: “Hide from screen sharing (best-effort)” with tooltip: “Works in many apps; not guaranteed. Use Share Safety Check.”
- “Share Safety Check” wizard: guided validation steps for each supported app you claim.

This aligns with Notchie’s own wording: “excluded from screen sharing in supported apps.” [4]

Telemetry/analytics plan (privacy-safe, opt-in)

Given competitor expectations (“no tracking, no data collection”), default should be **no telemetry**. Notchie’s App Store disclosure is “Data Not Collected,” and its privacy policy claims it does not collect, store, or transmit personal data. [67] Apple requires truthful App Privacy disclosures for whatever data you collect, including via third parties. [68]

Plan - Default: telemetry OFF.
- Opt-in: “Help improve the app” toggle that enables sending *anonymous, minimal* diagnostics: - app version + OS version, - feature flags used (mode, surface), - high-level performance counters (CPU spikes, frame drops), - error codes (speech backend errors, Deepgram reconnect count), - never raw script text, never audio, never transcripts. - Provide “Export Diagnostics” button that packages local logs, last session state, and crash traces for support without silent collection.

License and third-party dependency audit

Open-source licenses

- AutoPrompter is MIT-licensed. [69]
- Textream is MIT-licensed. [70]
- Notchprompt is MIT-licensed. [71]
  MIT permits reuse, modification, sublicensing, and selling, provided you include the copyright and license notice. [72]

Product requirement: ship an “Open Source Notices” screen and bundle all MIT license texts and attributions (AutoPrompter, Textream, Notchprompt) inside the app.

Apple frameworks / entitlements

AutoPrompter uses: - App sandbox + audio input entitlement + network client/server, plus user-selected file read/write and Keychain groups. [73]
- Local server (Network framework) for browser overlay. [74]

These must be re-audited for App Store / notarization acceptance depending on your distribution strategy.

Cloud services and terms

- Deepgram: streaming STT uses WebSocket, supports `KeepAlive` and `CloseStream`, and endpoint behavior is controlled by params like `endpointing` and `interim_results`. [75]

- OpenAI: AutoPrompter includes an LLM resync feature that calls a chat completions endpoint from the client. [46] OpenAI's safety guidance warns against deploying API keys in client-side apps and recommends routing through your own backend. [76] OpenAI also documents migrating from `POST /v1/chat/completions` to `POST /v1/responses`, which affects forward compatibility. [77]

Productization decision:
- MVP: ship with **local-only** speech features by default; treat Deepgram as a Pro/advanced opt-in where users bring their own key, or offer it only if you later build a backend proxy.
- Do not bake in your own OpenAI key in a desktop client; if you want a "built-in resync," build a minimal backend that holds the key and enforces rate limits.

Delivery roadmap and handoff

QA test matrix

Notchie explicitly claims being hidden on common meeting apps (Zoom/Meet/Teams in public pages), and your app must validate those claims for each supported OS version. [78]

Dimension	Test cases	Acceptance criteria
macOS versions	14.6, 15.x latest	No crashes; overlay renders correctly; permissions behave; entitlements consistent
Hardware	MacBook with notch; MacBook without notch; Intel + Apple Silicon	Notch mode degrades gracefully ("Pinned to top bar" on no-notch); performance acceptable
Displays	Single display; external display; Sidecar-like extended display behavior	Fixed-display + follow-mouse policies consistent; external teleprompter mode stable [79]
Audio devices	Built-in mic; AirPods/Bluetooth; USB mic; aggregate device; device hot-swap mid-session	No dead mic; engine restarts reliably (AutoPrompter already restarts on config change). [36]
Conferencing apps	Screen share + recording in: major meeting apps (verify invisibility)	Overlay excluded in "supported apps"; if not excluded, app surfaces "Not safe" warning and suggests workaround. [80]
Recording apps	OS screen recording, third-party recorders	Document behavior per recorder; never promise universal invisibility
Remote browser	iPhone Safari / Android Chrome / desktop browsers on LAN	10Hz updates; reconnect works; no memory leak in long sessions [34]
Imports	PPTX with notes; PPTX without notes; large decks	Pages extracted correctly; good errors ("No presenter notes found"). [81]
Accessibility	Large text; high contrast; VoiceOver basics	Core flows usable; no trapping focus

Animation and motion spec (timings and behaviors)

AutoPrompter's notch overlay already uses a two-phase expand + content fade animation (~0.4s expand, then ~0.25s content reveal, and ~0.3–0.4s dismiss). [82] The new app should formalize motion so it feels consistent across surfaces:

Overlay entry - Notch-pinned:
- Phase A: container expands from "notch size" → full panel (400ms, ease-out).
- Phase B: content fades/slides in (250ms, ease-out) after ~300–350ms delay.
- Floating: scale from 96% → 100% + opacity 0 → 1 (200ms).
- Fullscreen/external: fade in (200ms), keep content stable (no bouncing).

Highlight motion - Word-tracking highlight: animate highlight transition between words (80–120ms) with minimal blur.
- Voice-activated/classic: smooth-scroll interpolation (critical for perceived polish).

State transitions - Pause: subtle dim (opacity -15%) + "paused" indicator appear (150ms).
- Done: show "Done" state for 1.0s then auto-dismiss or countdown-to-next-page if enabled (AutoPrompter uses a "done" view and optional countdown). [83]

Monetization flow (14-day trial → lifetime Pro)

Entitlement model

States - `trialActive(untilDate)` - `trialExpired` - `proActive` - `graceOffline(daysRemaining)` (optional: 3–7 day offline grace to reduce support tickets)

Enforcement points - Pro-only features should be gated at: - Advanced modes: Word Tracking with Deepgram backend, remote browser mirroring controls, external display mirror mode, PPTX import (if you choose to make these Pro). Grounding: these are "complexity/value" features in AutoPrompter/Textream. [84]
- Presets/Profiles beyond a small free allowance (e.g., 1 free profile vs unlimited).

Paywall UX - Trigger conditions: - On first "Start" attempt: show "14 days free" modal with "Continue trial" and "Unlock Pro."
- On Pro-only action: sheet with feature explanation and one-click purchase. - Copy guidance (align with privacy expectations): - "Local-first. No accounts required." (If true in implementation; Notchie emphasizes no accounts and local operation.) [85]

Entitlement checks - At startup: validate receipt/license file. - At runtime: `EntitlementService.has(feature)` must be cheap and synchronous.

Roadmap with milestones and sprint-level tasks

Assume 2-week sprints; adjust to your team size.

Milestone	Scope	Key tasks (sprint-level)	Exit criteria
Foundation	Repo setup + refactor plan	Split AutoPrompter into modules; introduce AppCoordinator + Store + Entitlements skeleton; CI signing/ notarization pipeline (if direct distribution)	App builds; overlay can still run
MVP prompting	Notchie parity core	Script library + editor, voice-activated scroll, notch & floating overlays, shortcuts baseline, end-of-script behaviors	Trial users can complete a full call workflow
Share safety	Truthful invisibility	Implement privacy toggle + Share Safety Check wizard; build supported-app matrix and in-app “status” indicator	Documented behavior; no over-promising [86]
Pro import/ output	Differentiators	PPTX import; external display + mirror; multi-page session UI; presets/profiles	Pro features stable and demoable [87]
Remote browser	Secondary differentiator	BrowserServer hardening, QR connect, UI settings, LAN messaging, throttling	Remote mirror works on phone/ tablet [34]
Monetization	Trial → lifetime	Implement trial clock, paywall, purchase + restore flows, receipts	End-to-end conversion works
Speech accuracy	Premium polish	Word tracking engine tuning; mic switching reliability; optional Deepgram integration	Clear differentiation vs Notchie [88]
Ship	v1 launch	QA matrix execution; docs & onboarding; support tooling	Launch-ready build

Risk analysis and mitigations

Screen-share invisibility risk (highest):
- Risk: capture path ignores `sharingType = .none`; OS/app changes break invisibility. [6]
- Mitigation: best-effort messaging, explicit supported-app list per OS version, in-app validation wizard, and visual “unsafe” indicator.

Mic switching / audio engine instability:
- Risk: AVAudioEngine taps can stop working or not emit configuration notifications reliably in edge cases; user changes input mid-call. [89]
- Mitigation: proactive “mic health” watchdog (detect silent buffers), forced engine recreation (AutoPrompter already recreates engine on beginRecognition), and explicit mic selector UI. [44]

Cloud feature cost and key security:
- Risk: shipping cloud LLM/STT features can create variable costs and API key handling/security issues; OpenAI advises against embedding keys in client apps. [90]

- Mitigation: local-first default; “bring your own key” or backend proxy with strict limits; keep Pro value strong even without cloud.

App Store privacy/compliance risk (if you go MAS):
- Risk: telemetry or third-party SDKs force privacy disclosures; mismatch with “no data collected” expectations. [91]
- Mitigation: default no telemetry; opt-in only; clear privacy policy and disclosures.

Developer handoff checklist

- Source attribution and license notices included for MIT-licensed upstream repos and any third-party fonts/assets. [92]
- Entitlements and Info.plist strings audited:
- Microphone + Speech Recognition usage descriptions required; verify against Apple guidance. [61]
- Network client/server entitlements only if BrowserServer/Deepgram are enabled. [93]
- Architecture boundaries implemented as protocols/actors with tests for:
- PrompterEngine mode math (classic/voice/word tracking)
- SpeechSyncEngine event parsing (Deepgram JSON, Apple partials)
- BrowserServer broadcasting and reconnect
- “Share Safety Check” shipped with a maintained compatibility doc and test plan. [86]
- QA matrix executed across macOS 14.6 and macOS 15.x, notch/no-notch hardware, and common meeting/recording apps.
- Monetization:
- Trial clock correctness (time zones, offline usage, reinstall behavior)
- Purchase/restore flows fully tested
- Entitlement gating verified (no “Pro for free” loopholes)
- Support:
- User-facing diagnostics export
- Crash handling and log rotation
- Clear versioning and update strategy (AutoPrompter/Textream track GitHub releases; decide Sparkle vs MAS). [94]

---

[1] [5] [23] [31] [32] [69] [92] https://github.com/ComputelessComputer/autoprompter
https://github.com/ComputelessComputer/autoprompter

[2] [4] [7] [8] [14] [15] [18] [19] [20] [21] [24] [27] [29] [30] [35] [38] [67] [80] [85] https://apps.apple.com/sg/app/notchie/id6756745307
https://apps.apple.com/sg/app/notchie/id6756745307

[3] [9] [10] [11] [13] [70] [94] https://raw.githubusercontent.com/f/textream/master/README.md
https://raw.githubusercontent.com/f/textream/master/README.md

[6] [65] [86] https://developer.apple.com/documentation/appkit/nswindow/sharingtype-swift.enum/none
https://developer.apple.com/documentation/appkit/nswindow/sharingtype-swift.enum/none

[12] [16] [22] [25] [84] https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/README.md
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/README.md

17 https://www.notchie.app/privacy
https://www.notchie.app/privacy

26 raw.githubusercontent.com
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/DeepgramStreamer.swift

28 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/ExternalDisplayController.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/ExternalDisplayController.swift

33 39 81 87 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/PresentationNotesExtractor.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/PresentationNotesExtractor.swift

34 50 59 74 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/BrowserServer.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/BrowserServer.swift

36 44 48 53 63 88 raw.githubusercontent.com
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/SpeechRecognizer.swift

37 45 58 https://developers.deepgram.com/docs/audio-keep-alive
https://developers.deepgram.com/docs/audio-keep-alive

40 42 51 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/AutoPrompterService.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/AutoPrompterService.swift

41 43 54 56 57 64 82 83 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/NotchOverlayController.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/NotchOverlayController.swift

46 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/LLMResyncService.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/LLMResyncService.swift

47 76 90 https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety
https://help.openai.com/en/articles/5112595-best-practices-for-api-key-safety

49 71 https://raw.githubusercontent.com/saif0200/notchprompt/main/README.md
https://raw.githubusercontent.com/saif0200/notchprompt/main/README.md

52 79 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/NotchSettings.swift
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/NotchSettings.swift

55 75 https://developers.deepgram.com/reference/speech-to-text/listen-streaming
https://developers.deepgram.com/reference/speech-to-text/listen-streaming

60 raw.githubusercontent.com
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter.xcodeproj/project.pbxproj

61 https://developer.apple.com/library/archive/qa/qa1937/_index.html
https://developer.apple.com/library/archive/qa/qa1937/_index.html

62 78 https://notchie.macupdate.com/
https://notchie.macupdate.com/

66 Notchie - Invisible Mac Teleprompter for Screen Sharing
https://productdirs.com/projects/notchie?utm_source=chatgpt.com

68 91 https://developer.apple.com/app-store/app-privacy-details/
https://developer.apple.com/app-store/app-privacy-details/

72 https://opensource.org/license/mit
https://opensource.org/license/mit

73 93 https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/AutoPrompter.entitlements
https://raw.githubusercontent.com/ComputelessComputer/autoprompter/main/AutoPrompter/AutoPrompter/AutoPrompter.entitlements

77 https://developers.openai.com/api/docs/guides/migrate-to-responses/
https://developers.openai.com/api/docs/guides/migrate-to-responses/

89 https://stackoverflow.com/questions/78536093/swift-macos-avaudioengineconfigurationchangenotification-sometimes-not-getti
https://stackoverflow.com/questions/78536093/swift-macos-avaudioengineconfigurationchangenotification-sometimes-not-getti