AI Update and Platform Expansion Ideas

Development & writer is my name
Telegram Bot: `@Faryseneaidownloder_bot`

Document purpose: Proposal for AI-powered feature additions and platform expansion for the Farysene AI Telegram Bot. This document outlines goals, phased feature work, platform suggestions, UX/command changes, implementation guidance, config flags, and acceptance criteria for rollout.

Goals

- Add AI-powered features that increase usefulness (transcription, summaries, smart search, recommendations) while remaining performant and maintainable.
- Expand supported platforms using modular handlers; prefer `yt-dlp` where feasible for quick coverage.

Phase 1 — AI features (low–medium complexity)

1. Speech-to-text & subtitles

Description: Convert audio/video into text and produce subtitles (SRT/VTT) and lyric timing files (LRC).

Options: Local (Faster-Whisper) or hosted (Whisper API).

UX:

- New commands: `/transcribe` (upload file or link), `/subtitles` (video link).
- Auto-generate subtitles on supported links when user selects an inline keyboard option.

Tech notes:

- New module: `ai/transcribe.py` with `transcribe_to_srt(input_path, lang='auto')`.
- Cache transcripts by content hash; store SRT/VTT/LRC alongside media files.

2. Translation & TL;DR summaries

Description: Translate subtitles/text between `fa` and `en`; summarize long videos/threads.

UX: Inline buttons: `Translate to FA/EN`, `Summarize`.

Tech notes:

- New module: `ai/nlp.py` exporting `translate_text()` and `summarize_text()`.
- Backend can be API-based (LLM provider) or local models. Implement caching.

3. Smart search (linkless) for music & videos

Description: Natural-language search ("Find me song X", "Top lofi playlist").

Pipeline: Search providers (Spotify/YouTube) → re-ranker (heuristic or small model) → present results.

UX: `/search` returns 5–10 results with inline buttons to `Download` or `Play`.

Tech notes:

- New package: `search/search.py` with provider adapters and a reranker module.

4. Audio normalization & metadata tags

Description: Normalize loudness to -14 LUFS, write ID3 tags, embed cover art.

UX: Toggle option `Normalize audio` before sending a file.

Tech notes:

- Module: `audio/postprocess.py` using `pyloudnorm`, `pydub` and `mutagen`.
- Controlled via a config flag and inline button.

Phase 2 — AI features (medium–high complexity)

1. Karaoke / stems & timed lyrics

Description: Vocal/instrumental separation (Demucs/Open-Unmix) and align lyrics to timestamps.

UX: `Create karaoke version (instrumental + lyrics)` button after download.

Tech notes:

- Module: `audio/separation.py` unifying separation scripts with caching by content hash.

2. Recommendations & "similar to"

Description: Per-user embeddings and nearest-neighbor recommendations (FAISS).

UX: `More like this` button; personalized daily suggestions.

Tech notes:

- Module: `ai/recsys.py`; embeddings via `sentence-transformers`; store vectors and use `faiss-cpu`.
- Nightly batch refresh of user/item embeddings.

3. Thumbnail selection & caption generation

Description: Pick best video thumbnail frame and generate captions/hashtags.

UX: Buttons: `Best thumbnail`, `Generate caption/hashtags`.

Tech notes:

- `video/thumbnail.py` (frame scoring or CLIP-lite)
- `ai/captions.py` for LLM-based caption prompts and templates.

Phase 3 — Moderation & Operations

1. Moderation filters

Description: NSFW/unsafe classification and spam/abuse detection. Provide warnings or block content.

UX: Show warning messages or block with rationale; optionally allow admin override.

Tech notes:

- `ai/moderation.py` with lightweight classifier; rules configured in `.env` or a config file.

2. Predictive throttling

Description: Predict rate-limit risk for platforms and back off proactively.

Tech notes:

- Integrate EWMA counters and predictive heuristics into `task_manager`.
- Use historical error rates to adjust concurrency/backoff.

Platform expansion — suggested additions

Fast wins (supported by `yt-dlp`)

- Vimeo, Dailymotion, Reddit, Twitch clips (VOD support), Bilibili, Streamable, Imgur (GIF/video), OK.ru, VK videos, Rumble, Facebook (improvements via cookies)

Music/audio

- Bandcamp (metadata + free downloads only), Mixcloud, Audiomack, Deezer (metadata), Apple Music (metadata/previews)

Text exports

- Twitter/X threads → PDF/Markdown
- Reddit threads → PDF
- Medium/Substack → PDF

Implementation plan for new platforms

1. **Detector:** Add URL regexes in `handlers/detector.py`.
2. **Handler module per platform:** Prefer `yt-dlp` to fetch media; fallback to API or HTTP as needed.
3. **Normalize outputs:** Filename, metadata, thumbnails to a common convention.
4. **History & cache:** Call `add_download_history()` and use temp dirs with atomic rename to `downloads/`.
5. **Cookies/sessions:** Centralize cookie loading from a secure path (ignored by git).
6. **Rate limits & UA rotation:** Per-domain limits and user-agent rotation.
7. **Concurrency & safety:** Use per-job temp dirs, file/DB locks for cache hits.

Commands & UX additions (summary)

- `/transcribe` — produce subtitles; buttons for translate/summarize.
- `/search` — smart search with inline download buttons.
- After a successful download: inline buttons — `Normalize audio`, `Create karaoke`, `Best thumbnail`, `Generate caption`, `Export to PDF` (for text threads/posts).
- Admin: `/status` extended with per-domain rate and error counts.

Config, dependencies & optional extras

requirements.txt (core): `aiogram`, `APScheduler`, `requests`, `yt-dlp`, `beautifulsoup4`, `mutagen`, `pydub`, etc.

requirements-ai.txt (optional): `faster-whisper` or `openai`, `sentence-transformers`, `faiss-cpu`, `pyloudnorm`, `pillow`, `torch`, `demucs/openunmix` (optional heavy deps).

.env flags:

```
ENABLE_STT=1
ENABLE_RECSYS=0
ENABLE_KARAOKE=0
ENABLE_MODERATION=1
RATE_LIMITS=
PROXY=
GPU_AVAILABLE=0
```

Gate high-risk features behind `ALLOWED_USERS` or `PRIVATE_MODE`.

Security & policy

- Prefer official APIs when available and document ToS considerations in README.
- Gate downloads of potentially protected content; warn users about copyright risks.
- Store cookies and credentials securely and exclude them from the repository.

Acceptance criteria (rollout)

Phase 1:

- `/transcribe` successfully handles YouTube/TikTok links and uploaded audio/video up to configured length.
- `/search` returns 5–10 relevant results and routes to platform handlers.
- Audio normalization optional and metadata written; history saved.

Phase 2:

- Karaoke pipeline stable and cached; timed lyrics generated for English tracks.
- Recommendations produce sensible results for known artists.
- Thumbnail/caption generation works for major platforms.

Phase 3:

- Basic NSFW flagging and blocklist active.
- Rate-limit backoff reduces platform errors (measurable reduction in errors/events).

Optional scaffolding I can provide

If desired, the following stubs can be scaffolded and added to the repo:

- `ai/transcribe.py` (stubbed Faster-Whisper integration)
- `search/search.py` (Spotify/YouTube provider stubs)
- `handlers` for Vimeo, Dailymotion, Reddit using `yt-dlp`
- Inline button flows in `bot.py` for `/transcribe` and `/search`

Notes

This proposal is designed to be incremental: start with a conservative Phase 1 (low compute, high value), evaluate stability and UX, then expand into Phase 2 and Phase 3 where heavier compute and safety concerns are more relevant. Feature flags, caching, and per-user rate controls are recommended from the outset.

Contact

Development & Writing is my name
Telegram