Sign in

SenteLabsAI / **OpenExecutive** Public

Notifications | Fork 62 | Star 1.1k

Code | Issues 2 | Pull requests 11 | Discussions | Actions | Projects | Security

main | Go to file | Code

johnrufusone Merge pull requ... 63dd9c6 · 18 hours ago

.claude	Initial public release	2 months ago
.github	chore(deps): Bump act...	2 months ago
.vscode	Initial public release	2 months ago
docker	Initial public release	2 months ago
docs	Initial public release	2 months ago
evals	Initial public release	2 months ago
fixtures	Initial public release	2 months ago
packages	Merge pull request #6...	last month
scripts	Initial public release	2 months ago
.dockerignore	Initial public release	2 months ago
.env.example	feat: support local / se...	2 months ago
.gitignore	Initial public release	2 months ago
CHANGELOG.md	chore: open-source re...	2 months ago
CLAUDE.md	Initial public release	2 months ago
LICENSE	Initial public release	2 months ago
Makefile	Initial public release	2 months ago
README.md	docs: correct README ...	18 hours ago
SECURITY.md	chore: open-source re...	2 months ago
fly.api.qa.toml	Initial public release	2 months ago
fly.api.toml	Initial public release	2 months ago

About

AI-powered virtual executive team — a single coherent executive persona backed by 8 specialist Claude agents (FastAPI + Next.js).

openexec-ui-dev.fly.dev

ai anthropic claude fastapi llm multi-agent nextjs python rag typescript

Readme
License
Code of conduct
Contributing
Security policy
Activity
Custom properties
1.1k stars
12 watching
62 forks
Report repository

Releases

No releases published

Contributors 5

Languages

fly.honcho.toml	Initial public release	2 months ago
fly.ui.qa.toml	Initial public release	2 months ago
fly.ui.toml	Initial public release	2 months ago

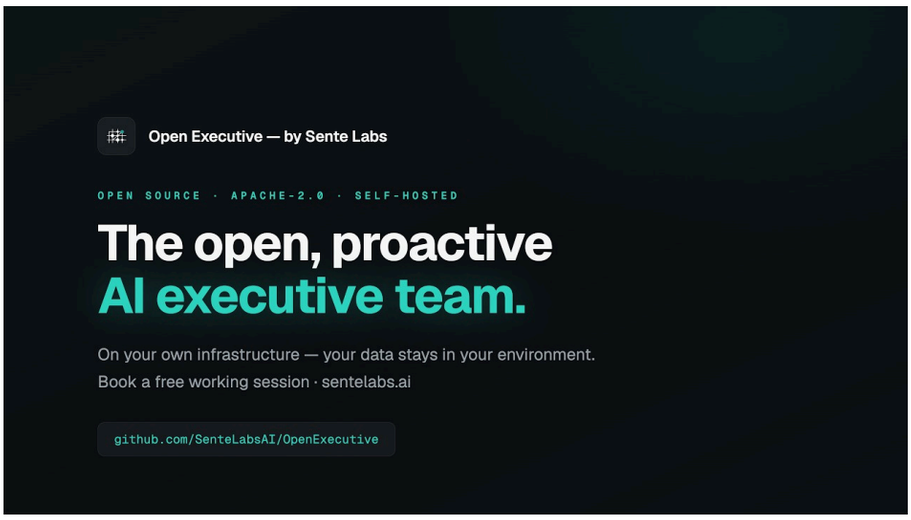
Python 82.2%
TypeScript 17.3% Shell 0.3%
Dockerfile 0.2% CSS 0%
Makefile 0%

Open Executive

CI passing | License Apache 2.0 | python 3.11+ | Next.js 15

An AI system that acts as your company's virtual executive team — a senior advisor with Harvard MBA-level knowledge, customized for your specific business.

Demo



A walkthrough of Open Executive in action — watch on YouTube.

What It Does

Developed by sentelabs.ai Open Executive provides a single coherent executive voice backed by eight specialist AI agents:

- **Chief Strategy Officer** — competitive analysis, M&A, market positioning, OKRs
- **Chief Financial Officer** — financial modeling, fundraising, unit economics, cash flow
- **Chief HR/People Officer** — hiring, compensation, performance, culture

- **General Counsel** — contracts, IP, employment law basics, compliance
- **Chief Operating Officer** — process design, vendor management, operational scaling
- **Chief Marketing Officer** — GTM strategy, brand, communications, PR
- **Chief Product Officer** — roadmap, prioritization, product strategy
- **Board Communications Director** — board decks, investor relations, governance

All responses come from one consistent executive voice. The internal agent architecture is never exposed to the user. Beyond Q&A, the system maintains episodic memory of past decisions and initiatives across sessions, and a built-in scheduler can proactively surface follow-ups and time-sensitive actions.

Architecture

```
User message
    ↓
Executive Orchestrator (claude-sonnet-4-6)
    ↓ tool use → parallel specialist calls
CSO / CFO / CHRO / GC / COO / CMO / CPO / Board
    ↓ each specialist retrieves relevant context
Built-in MBA knowledge + Your company documents
    ↓
Synthesized executive response
```

Knowledge — Two retrieval layers per specialist call: (1) built-in MBA-level Markdown (`knowledge/builtin/` , git-tracked) seeded into ChromaDB at startup, and (2) your uploaded company documents chunked and stored in a separate `company_docs` collection. RAG context is injected into the user turn, never the cached system prompt.

Episodic memory — After every response, a background `claude-haiku-4-5` pass extracts key decisions, initiatives, and advice into SQLite. The next session opens with a `<past_decisions>` block so the Executive remembers what it recommended last month.

Scheduler — A built-in job runner claims due actions via `UPDATE … RETURNING` to prevent double-firing. The API must run as a single instance; do not horizontally scale it without gating the scheduler first.

Prompt caching — The system prompt is structured so the Executive persona, company profile, and knowledge index are cached separately (up to 85% cache hit rate after the first few turns). No dynamic content ever goes in a cached block.

See docs/architecture.md for the full design.

Tech Stack

Layer	Choice
LLM backbone	Anthropic Claude API
Default model	`claude-sonnet-4-6` (Executive + most specialists)
Deep reasoning	`claude-opus-4-7` (CSO, CFO, GC, Board — with extended thinking)
Backend	Python 3.11 + FastAPI
Package manager	`uv`
Vector store	ChromaDB (local, embedded)
Episodic memory	SQLite
Web UI	Next.js 15 (App Router) + Tailwind
License	Apache 2.0

Repo Layout

```
openexecutive/
├── packages/
│   ├── core/
│   │   └── openexecutive/
│   │       ├── orchestrator/     # Executive pe
│   │       ├── agents/           # 8 specialist
│   │       ├── knowledge/        # ChromaDB sto
│   │       ├── memory/           # Company prof
│   │       ├── onboarding/       # Wizard state
│   │       ├── prompts/          # Persona + do
│   │       ├── api/              # FastAPI app
│   │       ├── integrations/     # Slack, Email
│   │       ├── scheduler/        # Background j
│   │       ├── alerts/           # Proactive al
│   │       ├── audit/            # Audit loggin
```

```
│   │       ├── architecture/     # Internal arcl
│   │       ├── workflows/        # Multi-step w
│   │       └── cli.py            # Click CLI
│   └── ui/                       # Next.js 15 w
├── evals/                        # Eval scenari
├── fixtures/                     # Demo company
├── scripts/                      # Operator scr
├── docker/                       # Dockerfile(s
├── fly.api.toml / fly.ui.toml    # Fly.io confi
├── fly.api.qa.toml / fly.ui.qa.toml  # Fly.io c
├── fly.honcho.toml               # Fly.io confi
└── docs/                         # Architecture
```

Quick Start

```
# Clone the repo
git clone https://github.com/SenteLabsAI/OpenExe
cd OpenExecutive

# Set your Anthropic API key
cp .env.example .env
# Edit .env and add ANTHROPIC_API_KEY=sk-ant-...

# Start everything
make dev
```

Open http://localhost:3000 to start chatting with your executive. The API runs on port 8000 and the UI on 3000.

> **First run:** requires Python 3.11+ and Node 22+. The initial `uv sync` pulls heavy ML dependencies (ChromaDB + sentence-transformers/PyTorch), and the first boot downloads a small embedding model (~90 MB) to build the local vector index — so the first `make dev` takes a few minutes before the app is ready. Subsequent starts are fast.

For contributors not using `make`:

```
cd packages/core
uv sync
source .venv/bin/activate
uvicorn openexecutive.api.main:app --reload --po

# In a second terminal
cd packages/ui && npm install && npm run dev
```

Run the Discord Bot

1. Create a Discord application at https://discord.com/developers/applications
2. Enable the **Message Content** privileged intent (Bot → Privileged Gateway Intents)
3. Invite the bot with `bot + applications.commands` scopes
4. Set env vars in `.env`: `DISCORD_BOT_TOKEN`, `DISCORD_APP_ID`, `DISCORD_GUILD_IDS`
5. Run the API normally — the bot starts as part of the FastAPI lifespan when `DISCORD_BOT_TOKEN` is set:

```
make dev
```

The bot is embedded in the API process (alongside the email poller, scheduler, and resumer) so it shares the same SQLite database and ChromaDB vector store under `/data` in production. Skip the token to disable.

For iterating on bot-only code without restarting the API, `make discord` runs the bot as a standalone process against the same local DB.

Users can DM the bot, `@mention` it in a channel (replies in a thread), or use `/ask` and `/today` slash commands. Slash commands sync to `DISCORD_GUILD_IDS` instantly on startup; leave blank for global registration (up to 1-hour propagation delay).

Deploying to production

Just set the secrets on the existing API app — no new Fly app required:

```
flyctl secrets set -a openexec-api-dev \
  DISCORD_BOT_TOKEN=... \
  DISCORD_APP_ID=... \
  DISCORD_GUILD_IDS=...
```

Discord user access is managed via the /people UI — add a Person row with `discord_user_id` set.

The machine restarts and the bot starts on the next lifespan boot. To disable in prod: `flyctl secrets unset -a openexec-api-dev DISCORD_BOT_TOKEN`.

Onboarding Your Company

The first time you visit the app, you'll be guided through a wizard to set up your company profile:

- Company basics (name, industry, stage, team size)
- Business model and revenue
- Competitive landscape
- Strategic priorities
- Culture and values
- Optional: financial position, document upload

After onboarding, the Executive will reference your specific company context in every response.

Interfaces

Interface	How to Use
Web UI	`http://localhost:3000`
Slack	Mention `@OpenExecutive` or DM the app
Email	CC or email the configured address (IMAP/SMTP poller)
Telegram	Message the configured bot
Google Chat	Mention the app in a space
Discord	DM the bot, `@mention` it in a channel, or use `/ask` / `/today` slash commands
CLI	`openexecutive chat`

Document Upload

Upload your pitch deck, financial model, strategy docs, or any company documents via the web UI or API. The Executive will reference them when relevant.

```
# Via CLI
openexecutive upload deck.pdf model.xlsx strategy

# Via API
curl -X POST http://localhost:8000/documents \
  -F "file=@deck.pdf" \
  -F "domain=strategy"
```

Deployment (Fly.io)

Two environments, each a separate set of Fly apps, driven by branch:

Environment	Trigger	Workflow
dev	push/merge to `main` (continuous)	`.github/workflows/deplo`
qa	push/merge to `qa` (deliberate promotion)	`.github/workflows/deplo` `qa.yml`

Both workflows use `dorny/paths-filter` to deploy only the changed app (API, UI, or both). QA is a stable twin of dev — same image and runtime, only the app name differs (`fly.api.qa.toml` / `fly.ui.qa.toml`) — so it lags `main` and stays vetted. An optional Honcho memory app (`fly.honcho.toml`) deploys independently.

Topology

App	Purpose	State
`openexec-api-{dev,qa}`	FastAPI + scheduler	Persistent volume `executive_data` at `/data`
`openexec-ui-{dev,qa}`	Next.js 15	Stateless
`openexec-honcho-dev`	Honcho per-person memory (optional)	Postgres-backed

> ⚠️ **Single-instance only**: The scheduler claims rows via `UPDATE … RETURNING` . Running two API machines would double-fire scheduled actions. `max_machines_running = 1` is set in `fly.api.toml` / `fly.api.qa.toml` — do not override it.

Required GitHub Actions secrets

Deploys authenticate with per-app Fly deploy tokens stored as repo (or org) Actions secrets. Generate each with `flyctl tokens create deploy -a <app> -x 999999h`:

Secret	App	Used by
`FLY_API_TOKEN_API`	`openexec-api-dev`	dev
`FLY_API_TOKEN_UI`	`openexec-ui-dev`	dev
`FLY_API_TOKEN_HONCHO`	`openexec-honcho-dev`	dev (honcho job)
`FLY_API_TOKEN_API_QA`	`openexec-api-qa`	qa
`FLY_API_TOKEN_UI_QA`	`openexec-ui-qa`	qa

Per-app runtime secrets (`ANTHROPIC_API_KEY`, `BACKEND_SHARED_SECRET`, the `AUTH_*` set, integration tokens) are set directly on each Fly app — see `scripts/fly-secrets.sh.example`.

One-time bootstrap (dev)

```
# 1. Create apps and volume
flyctl apps create openexec-api-dev
flyctl apps create openexec-ui-dev
flyctl volumes create executive_data --region iad

# 2. Set the required secret
flyctl secrets set -a openexec-api-dev ANTHROPIC_

# 3. Create deploy tokens and add as GitHub secre
flyctl tokens create deploy -a openexec-api-dev
flyctl tokens create deploy -a openexec-ui-dev

# 4. First deploy
gh workflow run "Deploy (dev)" -f target=both
```

QA bootstraps the same way against the `-qa` app names (push to the `qa` branch, or `gh workflow run "Deploy (qa)"`). See docs/deployment.md for the full runbook (operations, rollback, common failure modes, why `.flycast` isn't used).

Access control

The deployed UI is gated behind Google sign-in with an email allow-list, and the public API is protected by a shared-secret header between the UI proxy and the FastAPI backend. See docs/auth.md for the full setup (Google Cloud Console steps, required Fly secrets, adding/removing users, rotating secrets, and a debugging table).

Configuration

All settings via environment variables. Minimum required: `ANTHROPIC_API_KEY` — *unless* you configure a local or OpenRouter backend instead (see Running on Local Models). At least one provider must be set or the app refuses to start.

Variable	Required	
`ANTHROPIC_API_KEY`	Yes[1]	—
`DEFAULT_MODEL`	No	`claud`
`DEEP_REASONING_MODEL`	No	`claud`
`VECTOR_STORE_PATH`	No	`./chr`
`EPISODIC_DB_PATH`	No	`./epi`
`COMPANY_PROFILE_PATH`	No	`./com`
`ENABLE_CACHING`	No	`true`
`ROUTING_MODEL`	No	`claud` `20251`
`SLACK_BOT_TOKEN`	No	—
`SLACK_APP_TOKEN`	No	—
`EXEC_EMAIL_ADDRESS`	No	—
`EMAIL_POLL_INTERVAL_SECONDS`	No	`60`
`TELEGRAM_BOT_TOKEN`	No	—
`TELEGRAM_WEBHOOK_SECRET`	No	—
`DISCORD_BOT_TOKEN`	No	—

DISCORD_APP_ID	No	—
DISCORD_GUILD_IDS	No	—
DISCORD_NOTIFY_CHANNEL_ID	No	—
GOOGLE_CHAT_PROJECT_NUMBER	No	—
GOOGLE_CHAT_SERVICE_ACCOUNT_FILE	No	—
GOOGLE_OAUTH_CLIENT_ID	No	—
GOOGLE_OAUTH_CLIENT_SECRET	No	—
OPENROUTER_ENABLED	No	false
OPENROUTER_API_KEY	No	—
LOCAL_MODELS_ENABLED	No	false
LOCAL_BASE_URL	No	—
LOCAL_API_KEY	No	—
LOCAL_MODELS	No	—
LOCAL_TIMEOUT_S	No	300

`HONCHO_ENABLED`	No	`false`
`HONCHO_API_KEY`	No	—
`HONCHO_BASE_URL`	No	—

See .env.example for the full list.

> [1] `ANTHROPIC_API_KEY` is required only when you serve Claude models directly. It can be omitted entirely if you run on local models (`LOCAL_MODELS_ENABLED`) or route through OpenRouter (`OPENROUTER_ENABLED`).

Running on Local Models

Open Executive can run against any **OpenAI-compatible** local server — Ollama, LM Studio, vLLM, or llama.cpp — instead of (or alongside) the Anthropic API. Local model slugs route to your server through the same provider abstraction the hosted models use; no agent or orchestrator code changes.

```
# 1. Pull a capable, tool-use-friendly model (exa
ollama pull llama3.3

# 2. In .env — point at the local server and list
LOCAL_MODELS_ENABLED=true
LOCAL_BASE_URL=http://localhost:11434/v1   # Olla
```

```
LOCAL_MODELS=llama3.3

# 3. (Optional) run with NO Anthropic key — make
DEFAULT_MODEL=llama3.3
DEEP_REASONING_MODEL=llama3.3
ROUTING_MODEL=llama3.3
# ...and leave ANTHROPIC_API_KEY unset
```

The listed slugs appear in the **Council UI** model dropdown, so you can also run a hybrid setup — keep the Executive on Claude while flipping individual specialists to a local model per-agent.

Caveats. Server-side web search (`ENABLE_WEB_SEARCH`) and Anthropic prompt caching / extended thinking have no local equivalent and are automatically disabled for local models. Multi-agent routing leans heavily on tool use, so pick a model that's strong at it (e.g. Llama 3.3 70B, Qwen2.5) — small models may route poorly. `LOCAL_API_KEY` is only needed if your server (vLLM, or a gateway) requires a bearer token; Ollama and LM Studio need none.

Adding a New Specialist Agent

1. Create `packages/core/openexecutive/agents/your_agent.py` extending `BaseAgent`
2. Add a system prompt constant in `packages/core/openexecutive/prompts/domain_prompts.py`
3. Register in `packages/core/openexecutive/orchestrator/router.py` — add to `SPECIALIST_REGISTRY` and the `specialist` enum in `SPECIALIST_TOOLS`
4. Add domain alias to `DOMAIN_ALIASES` in `packages/core/openexecutive/knowledge/retriever.py`
5. Add knowledge docs to `knowledge/builtin/your_domain/`
6. Add at least 2 eval scenarios to `evals/scenarios/`
7. Submit a PR — CI requires all of the above

Development

```
make dev          # Start FastAPI + Next.js
make test         # Run Python tests
make eval         # Run eval suite
make lint         # Run ruff + mypy
make docker       # Build and run Docker stack

# Unit tests only (no API calls required)
pytest packages/core/tests/unit/ -v
```

Evaluation System

`evals/` contains 29 scenarios covering all 8 domains, scored by `claude-opus-4-7` as an LLM-as-judge. Each scenario defines a query, simulated company context, expected topics, required specialist routing, and a domain-specific rubric. Five scoring dimensions (persona coherence, domain accuracy, company context utilization, routing quality, actionability) are each rated 1–5. The CI gate requires ≥ 3.5/5 average; any dimension dropping > 10% vs `main` fails the PR.

Privacy

Everything in `company/` is gitignored — the profile YAML, uploaded documents, and the ChromaDB vector store. None of this leaves your local machine (or your own Fly volume in cloud deployments) except as part of prompts sent to the Anthropic API. Anthropic does not train on API data.

Contributing

See .github/CONTRIBUTING.md. All PRs must include:

- Working implementation (no stubs)
- Tests for new behavior
- Eval scenarios for new agents or prompt changes

License

Apache 2.0 — free to use commercially, requires attribution.