PRODUCT REQUIREMENTS DOCUMENT

HA AutoGen

Natural Language Automation & Dashboard Generator for Home Assistant

Version:	1.0
Date:	February 2026
Author:	David
Status:	Draft

1. Executive Summary

HA AutoGen is a Home Assistant add-on that enables users to create automations and Lovelace dashboards through natural language requests. It introspects the user's Home Assistant instance — entities, devices, areas, and existing configurations — then uses a large language model (LLM) to generate valid, deployable YAML from plain English descriptions.

The add-on targets Home Assistant enthusiasts who understand what they want their smart home to do but find the YAML configuration barrier frustrating or time-consuming. Rather than replacing manual configuration entirely, HA AutoGen acts as an intelligent scaffolding tool: it generates a starting point that users can review, refine, and deploy with confidence.

Core Value Proposition

Describe what you want in plain English. Review the generated YAML. Deploy with one click. Iterate naturally.

2. Problem Statement

Home Assistant is an extraordinarily powerful platform, but its power comes with complexity. Creating automations and dashboards requires understanding YAML syntax, knowing the correct service calls, entity ID formats, trigger/condition/action structures, and Lovelace card schemas. This creates several pain points:

- **Syntax friction:** Users know they want "turn off lights when no one is home" but translating that into correct YAML with the right trigger type, entity references, and service calls requires documentation lookup and trial-and-error.
- **Discovery gap:** Users often don't know what's possible. They may not realise they can combine a time condition with a presence trigger, or that a particular sensor can be used in a template.
- **Dashboard tedium:** Building Lovelace dashboards is particularly time-consuming. Creating a well-organised dashboard with the right cards, grouped by area, with conditional visibility — this is hours of manual YAML editing.
- **Iteration cost:** Modifying existing automations often means re-reading complex YAML, understanding the original intent, then making careful edits without breaking the structure.

The built-in automation editor has improved significantly but still requires users to think in terms of triggers, conditions, and actions rather than describing desired behaviour naturally.

3. Goals & Non-Goals

3.1 Goals

1. Enable users to describe automations and dashboards in natural language and receive valid, deployable YAML.
2. Provide full context awareness of the user's HA instance (entities, areas, devices, existing automations) so generated output references real entity IDs.
3. Implement a mandatory review step before any configuration is applied to the live system.
4. Support both local LLM inference (for privacy-conscious users) and cloud API backends (for convenience and capability).
5. Package as a standard Home Assistant add-on for easy installation and updates.
6. Generate output that follows Home Assistant best practices and is readable/maintainable by the user after generation.

3.2 Non-Goals (v1.0)

- Real-time voice control or conversational assistant (this is HA's Conversation agent domain).
- Replacing the HA automation editor — the goal is augmenting it with a generation layer.
- Managing integrations, add-ons, or system-level HA configuration.
- Supporting custom components or third-party frontend cards in v1.0.
- Multi-user collaboration or role-based access control.

4. Target Users

Persona	Profile	Key Need
Power User	Extensive HA setup, comfortable with YAML but wants to move faster. Dozens of automations already in place.	Speed up iteration. Generate boilerplate, then refine. Modify existing automations via natural language diffs.
Enthusiast	Moderate HA setup. Can follow documentation but struggles with complex YAML. Wants more sophisticated automations than the UI editor allows.	Bridge the knowledge gap. Describe intent, get working YAML with explanations of what each part does.
Newcomer	Recently set up HA. Has devices connected but hasn't created	Get started quickly. See what's possible with their existing entities.

	automations yet. Intimidated by YAML.	Learn by example from generated output.

5. System Architecture

5.1 High-Level Architecture

HA AutoGen runs as a Home Assistant add-on (Docker container) with the following components:

Component	Description
Web UI	Ingress-served frontend (accessible via HA sidebar). Provides the natural language input, YAML preview/editor, generation history, and deploy controls.
Context Engine	Connects to the HA Supervisor and WebSocket APIs to pull entity registry, device registry, area registry, service definitions, and existing automation/dashboard configs. Builds a structured context payload for the LLM.
LLM Interface	Abstraction layer supporting multiple backends: local models via Ollama/llama.cpp (HTTP API), Anthropic Claude API, or OpenAI-compatible APIs. Manages prompt construction, token budgets, and response parsing.
Validator	Validates generated YAML against HA schemas. Checks entity ID references against the live registry. Flags potential issues (e.g., referencing entities that don’t exist, deprecated service calls).
Reviewer	Analyses existing automations and dashboards using the LLM. Identifies inefficiencies, deprecated patterns, missing best practices, redundancies, and security concerns. Produces actionable improvement suggestions with ready-to-deploy replacement YAML.
Deployer	Applies approved configurations to the HA instance. For automations: writes to automations.yaml or uses the REST API. For dashboards: writes Lovelace YAML or uses the storage-based dashboard API. Handles backup and rollback.

5.2 Data Flow

7. User enters a natural language request in the Web UI.
8. Context Engine pulls current HA state and builds the context payload.
9. LLM Interface constructs the prompt (system prompt + context + user request) and sends to the configured backend.
10. LLM returns generated YAML with inline comments explaining each section.
11. Validator checks the YAML for structural correctness and entity reference validity.

12. User reviews the output in the Web UI's YAML editor (with syntax highlighting and diff view for modifications).
13. On approval, Deployer writes the configuration and triggers a reload.

5.3 Authentication & Access

The add-on authenticates to Home Assistant using the Supervisor API token (automatically provided to add-ons). The Web UI is served through HA's ingress proxy, inheriting the user's HA authentication session. No separate authentication layer is needed.

For cloud LLM backends, API keys are stored in the add-on's configuration (accessible through the HA add-on settings panel) and are never exposed to the frontend.

6. Functional Requirements

6.1 Context Gathering

The Context Engine must be able to retrieve and structure the following data from the Home Assistant instance:

Data Source	Information Retrieved	Purpose
Entity Registry	Entity IDs, friendly names, domains, device class, unit of measurement, current state	Enables LLM to reference correct entities and understand their capabilities
Area Registry	Area names, IDs, associated entities	Enables area-based grouping in dashboards and area-scoped automations
Device Registry	Device names, manufacturers, models, associated entities	Helps LLM understand entity relationships and device capabilities
Service Registry	Available services per domain, their parameters and descriptions	Ensures generated automations use valid service calls with correct parameters
Existing Automations	Current automation YAML configs	Context for modification requests; avoids generating duplicate automations
Existing Dashboards	Current Lovelace configurations	Context for dashboard modification and understanding current layout

6.2 Natural Language Processing

Request types the system must handle (with example inputs):

- **New automation:** “Turn off all lights in the living room when the TV is turned off after 10pm.”
- **Modify existing automation:** “Change my morning routine automation to also open the bedroom blinds.”
- **New dashboard:** “Create a dashboard showing all temperature and humidity sensors grouped by room.”
- **Dashboard modification:** “Add a row of media player cards to my main dashboard.”
- **Exploratory:** “What automations could I create with my motion sensors and lights?”
- **Troubleshooting:** “Why might my garage door automation not be triggering?” (analyse existing automation YAML)

6.3 YAML Generation

Generated YAML must:

- Be syntactically valid YAML that passes HA’s configuration check.
- Reference only entity IDs that exist in the user’s instance (validated against the live registry).
- Include inline comments explaining what each section does and why specific choices were made.
- Follow HA best practices (e.g., using device triggers over state triggers where appropriate, using areas for grouping).
- Be formatted consistently with proper indentation.

6.4 Validation Pipeline

Before presenting output to the user, all generated YAML passes through the validation pipeline:

Check	Description	Failure Behaviour
YAML Syntax	Parse YAML and verify structural validity	Block output; retry with LLM
Entity References	Cross-check all entity_id values against the live entity registry	Flag with warning; suggest corrections
Service Calls	Verify service domains and service names are valid	Flag with warning; suggest valid alternatives
Schema Compliance	Validate against HA automation/Lovelace JSON schemas	Block output; retry with LLM including error details
Duplicate Detection	Compare against existing automations for functional overlap	Warn user of potential duplicate

6.5 Review & Deploy

The review interface must provide:

- Syntax-highlighted YAML editor with the generated output pre-loaded.
- Diff view when modifying existing automations/dashboards (side-by-side old vs. new).
- Validation status indicators (pass/warning/fail for each check in the pipeline).
- One-click deploy that writes the YAML and triggers the appropriate HA reload (automation reload or Lovelace refresh).
- Automatic backup of the previous configuration before overwriting.
- Rollback capability to revert the last deployment.
- Generation history with the ability to revisit, re-edit, and re-deploy previous outputs.

6.6 Configuration Review & Analysis

The add-on provides an LLM-powered review mode that analyses the user's existing automations and dashboards, producing actionable improvement recommendations. This feature serves as a strong onboarding entry point — users get immediate value from the tool before generating anything new.

6.6.1 Automation Review

The system analyses existing automations against the following criteria:

Check Category	What It Detects	Example Finding
Trigger Efficiency	Polling-based triggers (time_pattern) where event-driven alternatives exist (state, device)	"Automation 'Motion Lights' polls every 10s. Switch to a state trigger on binary_sensor.hallway_motion for instant response and lower resource usage."
Missing Guards	Automations lacking time, presence, or state conditions that could cause unintended execution	"'Porch Light On' triggers on sunset but has no condition to check if anyone is home. Consider adding a person.* state condition."
Deprecated Patterns	Deprecated service calls, old-style entity references, platform syntax that has been superseded	"Service 'homeassistant.turn_on' can be replaced with domain-specific 'light.turn_on' for explicit control over brightness and colour."
Redundancy	Multiple automations with overlapping triggers and actions that could be consolidated	"Automations 'Kitchen Off Night' and 'Kitchen Off Away' both turn off kitchen lights. Consider merging with an 'or' condition block."
Security Concerns	Automations controlling sensitive domains (lock, alarm_control_panel, cover) without adequate safeguards	"'Auto Unlock' unlocks the front door when a specific device connects to WiFi. This is spoofable — consider adding a second factor like a presence zone condition."
Error Resilience	Automations that don't handle unavailable entities, missing	"If sensor.outdoor_temp becomes unavailable, 'Climate Control' will fail

	‘continue_on_error’ flags, or lack of ‘choose’ fallbacks	silently. Add a condition to check availability or a default fallback.”

6.6.2 Dashboard Review

Dashboard analysis covers layout, completeness, and usability:

- **Unused entity detection:** Identifies entities with no dashboard representation that likely should have one (e.g., climate entities, media players, security sensors not shown on any card).
- **Inconsistent card usage:** Flags cases where similar entity types use different card types without clear reason (e.g., some temperature sensors on gauge cards, others on entities cards).
- **Missing area coverage:** Detects areas with entities that have no corresponding dashboard view or section.
- **Card type recommendations:** Suggests more appropriate card types based on entity domain and data characteristics (e.g., history-graph for trending sensors, thermostat card for climate entities).
- **Layout optimisation:** Identifies overly long single-column layouts, underutilised horizontal-stack/vertical-stack groupings, and opportunities for conditional cards.

6.6.3 Review Output Format

Each review finding includes:

14. A severity level (Info, Suggestion, Warning, Critical) to help the user prioritise.
15. A plain-English explanation of the issue and why it matters.
16. The specific automation or dashboard section affected, with a link to view it.
17. A ready-to-deploy improved YAML snippet that the user can review and apply through the standard deploy pipeline.
18. A one-click “Apply Fix” button that loads the improved YAML into the diff editor for review before deployment.

Users can run a full review (all automations and dashboards) or a targeted review (single automation, single dashboard, or a specific area). The full review produces a summary report with findings grouped by severity, making it easy to tackle the most impactful improvements first.

Onboarding Value

Configuration Review is the recommended first action for new users. Install the add-on, run a full review, and immediately see how your existing setup can be improved — before generating anything new. This builds trust in the tool’s understanding of your system.

7. LLM Backend Configuration

The add-on supports multiple LLM backends to accommodate different user priorities around privacy, cost, and capability:

Backend	Pros	Cons	Minimum Model
Local (Ollama)	Full privacy; no API costs; no internet required	Requires significant hardware; slower; less capable on complex requests	Qwen 2.5 Coder 32B or DeepSeek Coder V2 (recommended: 70B class model for reliable output)
Anthropic API	Highest quality output; excellent at structured YAML generation; strong at following HA conventions	Requires internet; API costs; data leaves the network	Claude Sonnet 4.5 (recommended: Claude Opus 4.5 for complex multi-entity automations)
OpenAI-compat.	Flexible; supports many providers (OpenAI, Groq, Together, local vLLM)	Quality varies by provider/model	GPT-4 class or equivalent

Local LLM Note

For local deployment on AMD Strix Halo with 128GB unified memory, a 70B parameter model (e.g., Llama 3.1 70B, Qwen 2.5 72B) provides reliable YAML generation quality. The large unified memory pool eliminates the need for quantisation, preserving output quality for structured generation tasks.

7.1 Prompt Architecture

The LLM prompt is constructed in layers:

19. **System Prompt:** Defines the LLM's role, output format requirements, YAML conventions, and Home Assistant-specific rules (e.g., automation structure, valid trigger platforms, Lovelace card types).
20. **Context Payload:** Structured representation of the user's HA instance. Entities grouped by domain and area. Existing automations summarised. Token budget management ensures the most relevant entities are prioritised.
21. **User Request:** The natural language input, optionally enriched with clarifications from a follow-up dialogue.
22. **Output Schema:** Explicit instructions for the response format — valid YAML wrapped in code fences, with inline comments, and a brief explanation section.

7.2 Token Budget Management

Large HA instances may have hundreds or thousands of entities. The Context Engine must intelligently manage the token budget:

- Prioritise entities relevant to the request (e.g., if the user mentions “living room lights,” prioritise light.* entities in the Living Room area).
- Group entities by domain and area, providing counts for unprioritised groups (e.g., “47 additional sensor entities available — ask to include specific ones”).
- Cache the full entity list and serve a filtered subset per request.
- For local models with smaller context windows, implement more aggressive filtering with explicit fallback prompting (“if you need information about additional entities, say so”).

8. Technical Specifications

8.1 Technology Stack

Layer	Technology
Runtime	Python 3.12+ (Alpine-based Docker image for add-on)
Web Framework	FastAPI (backend API) + lightweight frontend (htmx or React, TBD)
HA Communication	Home Assistant WebSocket API (real-time) + REST API (CRUD operations)
YAML Processing	ruamel.yaml (round-trip parsing, comment preservation)
LLM Client	httpx (async HTTP client for all backends; no heavy SDK dependencies)
Validation	voluptuous (HA’s own schema validation library) + custom entity/service checkers
Data Storage	SQLite (generation history, backups, user preferences) stored in /data/ (persistent across updates)
Code Editor	CodeMirror 6 (YAML mode, syntax highlighting, diff view)

8.2 Add-on Configuration Schema

The add-on exposes the following configuration options through the HA add-on settings panel:

Option	Type	Default	Description
llm_backend	enum	ollama	LLM backend: ollama, anthropic, openai_compat
llm_api_url	string	http://localhost:11434	API endpoint URL
llm_api_key	password	(empty)	API key for cloud backends
llm_model	string	(auto)	Model identifier
auto_backup	boolean	true	Backup config before deploy

max_context_entities	integer	200	Max entities in LLM context

8.3 Add-on Manifest (config.yaml)

Key add-on manifest properties:

- homeassistant_api: true (enables Supervisor API access).
- ingress: true (serves Web UI through HA's authenticated ingress proxy).
- auth_api: true (validates HA user sessions).
- Persistent storage at /data/ for SQLite database and backup files.
- Map the HA config directory (read/write) for direct YAML file manipulation.

9. User Interface

9.1 Main View — Generation Interface

The primary interface is a clean, focused generation view:

- **Input area:** Large text input with placeholder examples. Supports multi-line descriptions. A mode toggle (Automation / Dashboard / Review / Ask) sets the generation context.
- **Context sidebar (collapsible):** Shows detected entities relevant to the current request. Allows the user to pin/unpin entities for context inclusion. Displays area groupings.
- **Output panel:** Syntax-highlighted YAML editor powered by CodeMirror. Validation status badges. Deploy and Rollback buttons.

9.2 Review View — Configuration Analysis

The review interface presents the results of automation and dashboard analysis:

- **Review launcher:** Options for Full Review (all automations and dashboards), Automations Only, Dashboards Only, or a targeted single-item review. A progress indicator during LLM analysis.
- **Findings list:** Grouped by severity (Critical, Warning, Suggestion, Info) with expandable detail cards. Each finding shows the affected automation/dashboard name, the issue, and the recommendation.
- **Apply Fix flow:** One-click loads the improved YAML into the diff editor (side-by-side with the current config), allowing the user to review changes before deploying through the standard pipeline.
- **Summary dashboard:** A health score for the overall configuration, with trend tracking across reviews to show improvement over time.

9.3 History View

A searchable, chronological list of all generations and reviews. Each entry shows the original request, generated type (automation/dashboard/review), timestamp, deployment status (deployed/draft/rolled-back), and a one-click option to re-open in the editor.

9.4 Settings View

LLM backend configuration (duplicated from the add-on settings for convenience, with a connection test button). Entity context preferences. Prompt customisation (advanced: lets users add custom instructions to the system prompt, e.g., “Always use MQTT triggers for Zigbee devices”).

10. Development Phases

Phase 1: Foundation (Weeks 1–3)

- Add-on scaffold: Docker image, config.yaml, ingress setup.
- Context Engine: Entity, area, device, and service registry retrieval via WebSocket API.
- LLM Interface: Ollama backend integration with basic prompt template.
- Minimal Web UI: Text input → raw YAML output (no editor, no validation).
- Deliverable: Generate a basic automation YAML from a natural language request using a local model.

Phase 2: Validation & Review (Weeks 4–6)

- Validation pipeline: YAML syntax, entity reference checking, service call validation.
- CodeMirror YAML editor with syntax highlighting.
- Retry logic: If validation fails, re-prompt the LLM with error context.
- Cloud backend support: Anthropic and OpenAI-compatible API integration.
- Deliverable: Validated automation generation with a proper review interface.

Phase 3: Deployment, History & Config Review (Weeks 7–10)

- Deploy engine: Write automations to automations.yaml and trigger reload.
- Backup system: Snapshot config before deployment; single-step rollback.
- Generation history: SQLite storage, history view, re-open and re-deploy.
- Diff view for modification requests (existing vs. proposed).
- Configuration Review engine: Automation analysis against best-practice rules (trigger efficiency, missing guards, deprecated patterns, redundancy, security).

- Review output UI: Severity-grouped findings with one-click “Apply Fix” loading into the diff editor.
- Deliverable: Full automation lifecycle plus existing config review and improvement suggestions.

Phase 4: Dashboards & Dashboard Review (Weeks 11–14)

- Lovelace YAML generation: Support for common card types (entities, gauge, history-graph, media-control, weather, etc.).
- Area-based dashboard generation (“Create a dashboard for all rooms”).
- Dashboard deploy via the Lovelace storage API.
- Dashboard review: Unused entity detection, inconsistent card usage, missing area coverage, card type recommendations, layout optimisation.
- Full review mode: Combined automation + dashboard analysis with summary report.
- Deliverable: Complete generation + review capabilities for both automations and dashboards.

Phase 5: Polish & Advanced Features (Weeks 15+)

- Exploratory mode: “What can I automate with my current setup?”
- Targeted review: Single-automation or single-area scoped analysis.
- Template support: User-defined prompt additions and preferred patterns.
- Token budget optimisation for large instances.
- Community prompt library (stretch goal).

11. Risks & Mitigations

Risk	Severity	Impact	Mitigation
LLM generates invalid YAML	Medium	User sees broken output; loses trust in the tool	Multi-layer validation pipeline. Automatic retry with error context. Block deployment of invalid YAML.
Hallucinated entity IDs	High	Automation references non-existent entities; fails silently at runtime	Mandatory entity registry cross-check. Fuzzy matching to suggest corrections. Clear validation warnings in the UI.
Destructive automation deployed	High	Automation causes unintended actions (e.g., unlocking doors, disabling alarms)	Mandatory review step. Automatic backup before deployment. One-click rollback. Flag security-sensitive entity domains

			(lock, alarm, cover) with extra warnings.
Token budget exceeded	Medium	Large instances overflow context window; LLM produces degraded output	Intelligent entity filtering by relevance. Progressive disclosure. Explicit token budget tracking.
Local model too slow	Low	Poor UX from long generation times	Streaming output display. Progress indicators. Recommend minimum hardware specs. Offer cloud fallback.
API key exposure	Medium	Cloud API keys compromised	Keys stored server-side only, never sent to frontend. Use HA's secrets management where possible.

12. Success Metrics

Metric	Target (v1.0)	Measurement
YAML validity rate	> 95% of generations pass validation on first attempt	Automated: validation pass/fail logging
Entity accuracy	> 98% of entity references are valid	Automated: entity cross-check results
Deploy rate	> 70% of generations are deployed (not abandoned)	Automated: deploy vs. discard ratio
Generation time (local 70B)	< 30 seconds for simple automations	Automated: request-to-output timing
Generation time (cloud API)	< 10 seconds for simple automations	Automated: request-to-output timing
Rollback usage	< 5% of deployments are rolled back	Automated: rollback event logging

13. Future Considerations

The following features are explicitly out of scope for v1.0 but should be considered in the architecture to avoid blocking future development:

- **Multi-turn conversation:** Allow the user to iteratively refine generated output through dialogue ("Make it also check if the sun has set").

- **Blueprint generation:** Generate HA Blueprints (parameterised automation templates) that can be shared.
- **Custom card support:** Extend dashboard generation to support popular custom cards (Mushroom, mini-graph-card, etc.).
- **Batch generation:** “Set up automations for all my rooms to turn off lights after 30 minutes of no motion.”
- **Integration with Assist:** Allow generation requests via HA’s voice assistant pipeline.
- **HACS distribution:** Publish the add-on through the Home Assistant Community Store for broader reach.

14. Appendix: Example Prompt Template

Below is a simplified version of the system prompt structure. The actual prompt will be more detailed and include HA-specific YAML schemas.

System Prompt (Abbreviated)

You are HA AutoGen, an expert Home Assistant configuration generator. You generate valid Home Assistant YAML from natural language descriptions. Rules: • Only reference entity IDs from the provided entity list. • Use the correct automation schema: trigger, condition (optional), action. • Include inline YAML comments explaining each section. • Wrap output in ```yaml code fences. • If the request is ambiguous, ask a clarifying question before generating. • Prefer device triggers over state triggers where applicable. • Use areas for grouping when the user references rooms. [ENTITY CONTEXT INSERTED HERE] [USER REQUEST INSERTED HERE]

End of Document