DSL VECTOR DATABASE

Domain Specific Languages

Chroma Vector Database Structure

Technical Documentation

Overview

This document describes the structure and contents of the Chroma vector database used in this project.

Database Type

Chroma DB - An open-source embedding database designed for storing and querying vector embeddings alongside metadata.

Chroma is specifically built for AI applications, enabling efficient semantic search and retrieval-augmented generation (RAG) workflows.

Domain Specific Language (DSL)

Technology

TextX - A Python framework for building Domain-Specific Languages (DSLs). TextX enables the creation of custom languages with defined grammars and model transformations.

Purpose

This database stores TextX-based DSLs primarily sourced from GitHub repositories. Each DSL includes its complete documentation, grammar definition, sample models, and analytical metadata for both grammar complexity and model coverage.

File Structure per DSL

Each DSL collection contains the following file types:

- **Documentation:** .md or .rst files containing DSL documentation and usage guides
- **Grammar Files:** .tx files defining the syntax and structure of the DSL
- **Model Files:** Files with DSL-specific extensions containing example models/programs written in the DSL, more models both valid and invalid added to improve the rules coverage metric
- **Grammar Analysis:** .txt files containing metrics about grammar complexity, including rule counts, dependencies, and structural analysis
- **Model Metadata:** .metadata files (one per model) analyzing which grammar rules each model uses and coverage statistics
- Array of all grammar rules.

Collections Structure

The database contains collections for different TextX DSLs, each sourced from GitHub repositories. Below is the template structure for documenting each DSL collection:

DSL Collection Template

Property	Details
DSL Name	[Name of the DSL]
Description	[Brief description of what this DSL is used for]
Source	[GitHub repository URL]
File Extension	[Extension used for model files, e.g., .robot, .entity, .workflow]
Document Count	[Total number of embedded documents for this DSL]
Grammar Files	[Number of .tx grammar files]
Model Files	[Number of example model files]
Synthetic models	[Number of synthetic model files and whether they are valid or invalid]
Documentation	[Number of .md/.rst documentation files]

Grammar Analysis Metrics:

Each grammar file (.tx) has an associated .txt analysis file containing:

- Total number of grammar rules
- Rule complexity metrics

Model Metadata Analysis:

Each model file has an associated .metadata file containing:

- Which grammar rules are used in the model
- Grammar coverage percentage

DSL Collection 1: Entity DSL

Property	Details
DSL Name	Entity DSL

Description	A simple DSL for modeling data. This example demonstrates how to generate source code from the model and is used for the Entity tutorial in TextX documentation.
Source	GitHub (TextX community examples)
File Extension	.ent
Grammar Files	1 .tx file (entity.tx)
Model Files	1 .ent model file
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

DSL Collection 2: IBM Rhapsody DSL

Property	Details
DSL Name	IBM Rhapsody DSL
Description	An example of loading a model made using IBM Rational Rhapsody, demonstrating TextX's ability to work with models from external tools.
Source	GitHub (TextX community examples)
File Extension	.rpy
Grammar Files	1 .tx file (ibm_rapshody.tx)
Model Files	1 .rpy model file
Synthetic models	1 invalid 1 valid
Documentation	1 .md file

Use Cases:

- Model-driven development for embedded systems
- UML-based system architecture design
- Automated code generation from models

DSL Collection 3: State Machine DSL

Property	Details
DSL Name	State Machine DSL
Description	State machine language and example from Fowler's DSL book. Demonstrates source code generation (generating .dot file) and state machine interpreter.
Source	GitHub (TextX community examples)
File Extension	.sm
Grammar Files	1 .tx file (state_machine.tx)
Model Files	2 .sm model files
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

Note: *There is no grammar analysis. There was an unexpected error.*

DSL Collection 4: Hello World DSL

Property	Details
DSL Name	Hello World DSL
Description	A very simple language used for the hello world tutorial. This is the introductory example in TextX documentation for teaching basic DSL concepts.
Source	GitHub (TextX community examples)
File Extension	.hello
Grammar Files	1 .tx file (hello-world.tx)
Model Files	1 .hello model file
Documentation	1 .md file

Note: *This is the canonical first example from TextX official tutorials for learning DSL fundamentals. There are no invalid or extra valid models as the dsl is very simple.*

DSL Collection 5: JSON DSL

Property	Details

DSL Name	JSON DSL
Description	An example of using TextX for parsing data-interchange formats. Demonstrates how to implement parsers for well-known formats using TextX.
Source	GitHub (TextX community examples)
File Extension	.json
Grammar Files	1 .tx file (json.tx)
Model Files	5 .json model files
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

Key Features:

- Implements standard JSON specification
- Supports objects, arrays, strings, numbers, booleans, and null
- Useful for custom JSON processing and validation

DSL Collection 6: pyFlies DSL

Property	Details
DSL Name	pyFlies DSL
Description	An example of a real DSL for the description of cognitive reaction-time experiments. See the pyFlies project for more information.
Source	GitHub (TextX community examples)
File Extension	.pf
Grammar Files	1 .tx file (pyflies.tx)
Model Files	6 instances (.pf files) with metadata; Note: 1 is fully analyzed, others are dedicated test cases
Synthetic models	2 invalid
Documentation	1 .md file

Special Note: *The metadata analysis shows that one model is comprehensively analyzed while others serve as dedicated test cases for specific features.*

DSL Collection 7: Robot DSL

Property	Details
DSL Name	Robot DSL
Description	A simple DSL for moving a robot on an imaginary grid. This example demonstrates how to do model interpretation and is used for the robot tutorial in TextX documentation.
Source	GitHub (TextX community examples)
File Extension	.rbt
Grammar Files	1 .tx file (robot.tx)
Model Files	1 .rbt model file
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

Note: This DSL demonstrates model interpretation techniques - a key concept in TextX where models are executed/interpreted rather than just parsed.

DSL Collection 8: Workflow DSL

Property	Details
DSL Name	Workflow DSL
Description	A simple DSL for workflow descriptions, as mentioned in TextX example languages.
Source	GitHub (TextX community examples)
File Extension	.wf
Grammar Files	1 .tx file (workflow.tx)
Model Files	1 .wf model file
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

DSL Collection 9: Recipe DSL

Property	Details
DSL Name	Recipe DSL
Description	A modular DSL for defining cooking recipes with structured components. Supports recipe composition, ingredient specification, configuration, cooking plans, and complete recipe definitions.
Source	GitHub (Recipe DSL project)
File Extension	Various: .config, .ingredient, .plan, .recipe
Grammar Files	5 modular .tx files: • Base.tx (core definitions) • Config.tx (configuration rules) • Ingredient.tx (ingredient specifications) • Plan.tx (cooking steps) • Recipe.tx (complete recipe structure) Final grammar: recipec_combined.tx
Model Files	Multiple (16) recipe examples across different file types: 1 .config, 11 .ingredient, 1 .plan, 3 .recipe
Documentation	1 .md file

Key Features:

- Modular grammar design with 5 interconnected grammar files
- Comprehensive metadata coverage with 16 analyzed models
- Supports structured recipe composition from basic ingredients to complete recipes

DSL Collection 10: OCL-Python (Beuchey)

Property	Details
DSL Name	OCL-Python (Beuchey)
Description	It is a domain-specific language that translates OCL (Object Constraint Language) expressions into equivalent Python code and provides a wrapper library to simulate OCL behavior directly in Python.
Source	GitHub (Beuchey/ocl-python)
File Extension	.ocl
Grammar Files	1 .tx file (oclGrammar.tx)
Model Files	1 .ocl file

Synthetic models	1 invalid 1 valid
Documentation	1 .md file

Note: Both ‘UnaryOperator’ and ‘CollectionKind’ rules are duplicates. The identical rules have been removed.

DSL Collection 11: py-tabs (E2Music)

Property	Details
DSL Name	py-tabs (E2Music)
Description	It is a domain-specific language for describing and playing musical compositions using simplified notation, supporting guitar and keyboard tabs and chords, with a Python-based interpreter and GUI that generate and play music through FluidSynth.
Source	GitHub (E2Music/pyTabs)
File Extension	.song
Grammar Files	5 modular .tx files: • chords.tx • composition.tx • guitar-tab-note.tx • keyboard-tab-note.tx • tablature.tx Final grammar: py-tabs_combined.tx
Model Files	2 .song files
Synthetic models	2 invalid
Documentation	1 .md file

Note: Highly modular music DSL with separate grammars for different instrument types and composition elements.
The combined grammar contains the ‘Note’ rule twice.

DSL Collection 12: Silvera (alensuljkanovic)

Property	Details
DSL Name	Silvera

Description	It is a declarative DSL for modeling microservice architectures, built with textX, that supports multi-language code generation via plugins, evaluates architectures using microservice-specific metrics, and automatically generates documentation—all fully implemented in Python.
Source	GitHub (alensuljkanovic/silvera)
File Extension	.si
Grammar Files	1 .tx file (silvera.tx)
Model Files	33 .si model files
Synthetic models	2 invalid
Documentation	7 .md documentation files

Note: Silvera has one of the largest model collections with 33 examples, providing extensive coverage and use cases.

DSL Collection 13: AppLang (kosanmil)

Property	Details
DSL Name	AppLang
Description	Applang (APPlication LANGuage) is a domain specific language (DSL) for specification and rapid development of applications for mobile devices. Currently it generates source code for Android applications, which are runnable upon generation. iOS and Windows Phone applications are planned for the future.
Source	GitHub (kosanmil/applang)
File Extension	.alang
Grammar Files	1 .tx file (applang.tx)
Model Files	1 .alang model file
Documentation	1 .rst file

Note: The metadata for this model file was created manually. No synthetic models were created due to error:

❌ Error loading grammar: b'Can\'t use bool assignment inside repetition in rule "Platforms" at (22, 1).'

DSL Collection 14: Questionnaire (textX)

Property	Details
DSL Name	Questionnaire DSL
Description	It is a simple domain-specific language for defining interactive questionnaires, supporting conditional questions, multiple-choice or free-form answers with regex validation, and a terminal-based interpreter that runs the questionnaire and collects user responses.
Source	GitHub (textX/textx-lang-questionnaire)
File Extension	.que
Grammar Files	1 .tx file (questionnaire.tx)
Model Files	1 .que model file
Synthetic models	1 invalid 2 valid
Documentation	1 .md file

DSL Collection 15: Entity-Relationship (textX-lang-er)

Property	Details
DSL Name	Entity-Relationship DSL
Description	It is a DSL built with textX for specifying an application's entity-relationship data model, enabling automatic generation of full CRUDS functionality and planned support for multi-platform app code generation.
Source	GitHub (textX/textx-lang-er)
File Extension	.er
Grammar Files	1 .tx file (er.tx)
Model Files	1 .er model file
Documentation	1 .md file

Note: The metadata for this model file was created manually. No synthetic models were created due to:

❌ Error loading grammar: b'Can\'t use bool assignment inside repetition in rule "ConstraintType" at (95, 1).'

DSL Collection 16: BibTeX (textx-bibtex)

Property	Details
DSL Name	BibTeX DSL
Description	It is a DSL implemented with textX that provides a lightweight, easily extensible BibTeX parser for Python, turning `.bib` files into structured Python object models for simple processing and customization.
Source	GitHub (igordejanovic/textx-bibtex)
File Extension	.bib
Grammar Files	1 .tx file (biblex.tx)
Model Files	1 .bib model file
Synthetic models	2 invalid
Documentation	1 .md file

DSL Collection 17: ODP-EL (Open Digital Policy Expression Language)

Property	Details
DSL Name	ODP-EL (Open Digital Policy Expression Language)
Description	It is a textX-based implementation of a subset of the ODP-EL language, providing an early-stage framework for modeling and generating meta-model diagrams using PlantUML
Source	GitHub (igordejanovic/ODP-EL-textX)
File Extension	.odpl
Grammar Files	2 .tx files (odpel.tx, odpolicy.tx) Final grammar: odpel-combined.tx
Model Files	1 .odpl model file

Synthetic models	2 invalid
Documentation	1 .md file

Note: Both ‘EnterpriseObject’ and ‘Event’ rules are duplicates. The identical rules have been removed. Possible edge cases on grammar being incomplete. None of the models are valid.

DSL Collection 18: UsyML (textx)

Property	Details
DSL Name	UsyML
Description	It is a minimal SysML implementation built with textX in Python, enabling model parsing and generation of textual or graphical representations of SysML models through built-in textX generators.
Source	GitHub (igordejanovic/usyml-textx)
File Extension	.usyml
Grammar Files	1 .tx file (usyml.tx)
Model Files	9 .usyml models found on other repository
Synthetic models	2 invalid
Documentation	1 .md file

Note: Not all examples may be on the same TextX version; compatibility may vary.

DSL Collection 19: DFlow (ISSEL Lab)

Property	Details
DSL Name	DFlow
Description	It is a domain-specific language for defining dialogue flows for virtual assistants in smart environments, capable of automatically generating complete Rasa models for conversational AI deployment.
Source	ISSEL Lab, Aristotle University of Thessaloniki
File Extension	.dflow

Grammar Files	6 .tx files Final grammar: dflow_combined.tx
Model Files	29 .dflow files
Synthetic models	2 invalid
Documentation	2 .md files (1 general, 1 related to examples)

DSL Collection 20: GoalDSL (ISSEL Lab)

Property	Details
DSL Name	GoalDSL
Description	A goal-oriented requirements engineering DSL from ISSEL Lab at Aristotle University. Used for specifying goals, actors, dependencies, and requirements in software systems.
Source	ISSEL Lab, Aristotle University of Thessaloniki
File Extension	.goal
Grammar Files	12 .tx files Final grammar: goal_dsl_combined.tx
Model Files	29 .goal models
Synthetic models	2 invalid
Documentation	1 .md file

DSL Collection 21: Beaver DSL (ISSEL Lab)

Property	Details
DSL Name	Beaver DSL
Description	Beaver is a DSL for machine learning in live data. Like a beaver builds a dam, Beaver builds a pipeline for processing data in real-time.
Source	ISSEL Lab, Aristotle University of Thessaloniki

File Extension	.bvr
Grammar Files	5 .tx files Final grammar: Beaver_combined.tx
Model Files	5 .bvr model files
Synthetic models	2 invalid
Documentation	1 .md file

DSL Collection 22: sm-auto (ISSEL Lab)

Property	Details
DSL Name	sm-auto
Description	SmAuto is a DSL that enables users to program complex automation scenarios, for connected IoT devices in smart environments, that go beyond simple tasks.
Source	ISSEL Lab, Aristotle University of Thessaloniki
File Extension	.auto
Grammar Files	7 .tx files Final grammar: smauto_combined.tx
Model Files	11 .auto model file
Synthetic models	2 invalid
Documentation	1 .md file

Note: Both ‘NID’ and ‘FQN’ rules are duplicates. The identical rules have been removed.

DSL Collection 23: demol (ISSEL Lab)

Property	Details
DSL Name	demol

Description	Device Modeling Language (DeMoL) is a DSL designed for the automated synthesis of Internet of Things (IoT) device open-source software, in a hardware-aware manner.
Source	ISSEL Lab, Aristotle University of Thessaloniki
File Extension	.dev
Grammar Files	5 .tx files
Model Files	17 .dev model file
Synthetic models	2 invalid
Documentation	3 .md file

Note: There is no valid metamodel.

DSL Collection 24: codintxt (ISSEL Lab)

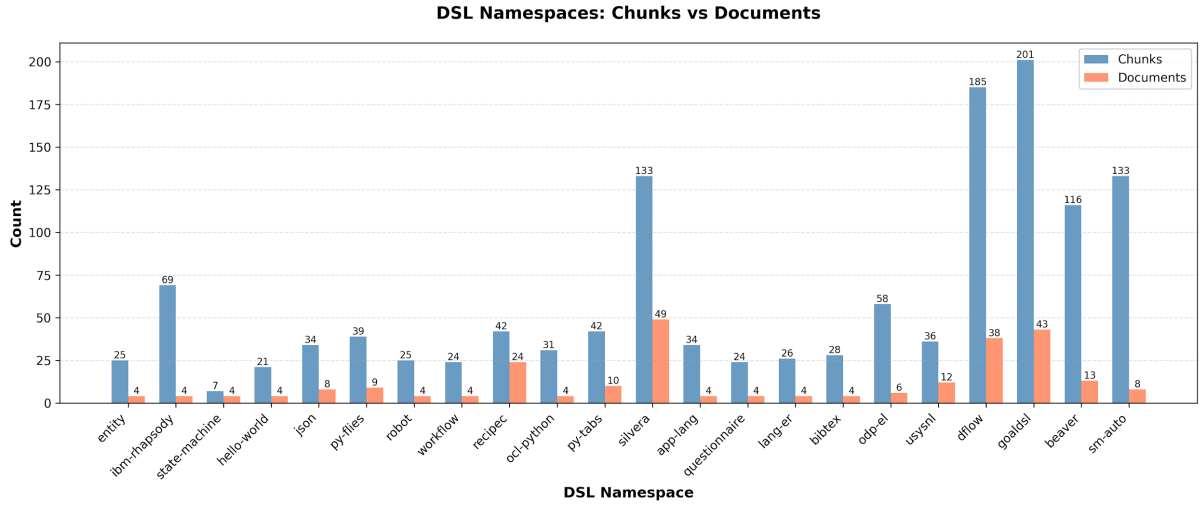
Property	Details
DSL Name	codintxt
Description	CodinTxt is a textual DSL for automating the development process for the Codin low-code platform .
Source	ISSEL Lab, Aristotle University of Thessaloniki
File Extension	.codin
Grammar Files	4 .tx files
Model Files	4 .codin model file
Synthetic models	2 invalid
Documentation	2 .md file

Note:

Statistics

Dsl Name	MFI	MFO	# of rules	Cov. of models
Entity	2	2	6	100%
IBM Rhapsody	2	5	5	100%
State Machine	?	?	8	75%
Hello World	1	1	2	100%
Json	2	5	5	100%
PyFlies	8	5	45	53.33%
Robot	1	2	6	100%
Workflow	1	4	4	100%
Recipe	1	5	15	100%
OCL Python	7	5	36	75%
Py-tabs	1	5	28	78.57%
Silvera	10	7	67	62.69%
App Lang	1	6	12	100%
Questionnaire	2	3	11	81.82%
Entity-Relationship	8	6	21	80.95%
BibTex	2	5	16	31.25%
ODP-EL	9	7	51	33.34%
Usyml	6	6	18	55.56%
Dflow	9	10	84	69.05%
GoalDsl	20	9	119	38.65%
Beaver	7	27	118	30.51%
Smauto	9	7	103	58.25%
DeMoL	5	6	94	?
Codin	1	5	78	25.64%

- **Max Fan In (MFI):** Maximum number of rules that reference a single non-terminal. This metric measures the maximum number of rules that reference a single non-terminal.
- **Max Fan Out (MFO):** Maximum number of unique non-terminals referenced by a single rule. This metric measures the maximum number of unique non-terminals referenced by a single rule.
- **# Of Rules:** Total number of grammar rules defined. This metric measures the total number of rules in the grammar.
- **Coverage of models:** The percentage of total rules used across all models.



Total namespaces: 24
Total chunks: 1658
Total documents: 386
Average chunks per namespace: 69.08
Average documents per namespace: 16.08

Database Structure

The database is organized as a **single Chroma collection (dsl_knowledge)** that stores all DSL-related content in a unified embedding space. This design enables **cross-DSL semantic retrieval**, allowing concepts shared between different metamodels or instances to reinforce each other semantically. To maintain precision when needed, the system supports **metadata-based filtering**, which can restrict searches to specific DSLs, file types, or namespaces during structured queries.

During ingestion, the pipeline performs **type-aware preprocessing and chunking**. Each file type (e.g., grammar, documentation, DSL instances, grammar analysis) is handled using an appropriate text splitter to optimize chunk coherence. Every resulting chunk is stored as an independent **document in Chroma**, carrying metadata such as source path, type, chunk index, and model metadata if applicable. In particular, **metamodels (.json files)** are automatically paired and serialized with

their corresponding **instance files**, ensuring that semantic and structural information remain linked throughout retrieval.

The database supports **two complementary retrieval modes**.

1. **Structured retrieval**, which leverages metadata filtering for precise, schema-aware lookups (e.g., by DSL, type, or chunk strategy).

2. **Semantic search (RAG retrieval)**, which performs vector-based similarity search to surface the most contextually relevant chunks for a given query.

This dual approach enables both **fine-grained control** over specific DSL data and **flexible, high-level reasoning** across the entire knowledge base.

Metadata Structure

Common Fields (All Files)

Field	Description
namespace	DSL identifier/name (added during chunking).
source_path	Absolute path to the file on disk.
filename	Name of the file with extension.
file_extension	File extension (e.g., `.tx`, `.md`, `.custom_instance`).
type	File category: `grammar`, `documentation`, `instances`, `granal`, or `other`.
created_date	File creation timestamp (ISO format).
modified_date	File last modification timestamp (ISO format).
version	Version number extracted from file content or filename.
chunk_index	Position of this chunk within the source document (added during chunking).
total_chunks	Total number of chunks created from this document (added during chunking).
chunk_strategy	Text splitting method used: `markdown`, `recursive`, or `default` (added during chunking).
overlap_size	Overlap size (integer) is used to correctly reconstruct the files.

Grammar-Specific Fields *(only for `.tx` files)*

Field	Description
has_many_grammars	Boolean indicating if DSL has multiple grammar files (modular structure).
is_final	Boolean indicating if this is the combined/final grammar (filename contains “combined”).

Instance-Specific Fields (only for model files)

Field	Description
total_rules_defined	Total number of rules defined in the DSL grammar.
rules_used_in_model	Number of grammar rules actually used in this model instance.
percentage_used	Percentage of grammar coverage (`(rules_used / total_rules) * 100`).
used_rules	JSON array of specific rule names used in this model (stored as string).
synthetic_instance	Indicates whether the model has been created later and wasn't part of the initial repo/commit/
is_valid	Indicates if the model can be validated successfully
validation_error	In case of the model being invalid, the error message is stored under this metadata.

Retrieval methods

The retrieval methods provide a comprehensive way to search, inspect, and reconstruct DSL-related data stored in the database. They support semantic queries (optionally with similarity scores), listing all available DSL namespaces, retrieving raw chunks from a specific namespace, and rebuilding full files from those chunks. Depending on the requested file type—**granal**, **documentation**, **instances**, **other**, or **grammar**—the methods can return either the stored segments or complete reconstructed files, handling chunk ordering, overlap resolution, and deduplication.

These retrieval capabilities work across all DSLs (such as Goal DSL, Recipe DSL, and Py-tabs) and allow consistent access to every stored resource regardless of its category.

Last Updated: 10/02/2026