

Using LLMs to extract knowledge and construct Knowledge Graphs

Rafael Brondani Prestes

Motivation

- LLMs often **hallucinate** or introduce **biases** in their responses, compromising the reliability of the generated information.
- Outputs are **highly dependent on prompt formulation**, requiring trial and error for accurate results.
- Responses from LLMs are frequently **overly generic**, lacking **specific details** and **verifiable references**, which limits their applicability in research or critical domains.
- Nowadays, it's unclear **which model performs best** in a given context or task.
- Building a **Knowledge Graph Manually** is a difficult and time consuming task.
- Extracting structured knowledge from **unstructured sources** like manuscripts and historical texts is **challenging**, especially due to variations in format, language, and semantics.

Objectives

- Identify the **most effective approach** to extract structured knowledge from structured data sources.
- Design and implement a pipeline to **build a Knowledge Graph** by **leveraging the power of LLMs**, combining their generative capabilities with structured reasoning.

Book of recipes and case studied

- Every recipe used in this project will be extracted from a PDF called “25 Ingredients 50 Meals”.
- To minimize the margin for error, a simple recipe was chosen:

RECIPE

Salmon & Potato Dish

Ready in **30 minutes in the oven**

8 minutes in the microwave

Serves **2-3 people**

Ingredients

- 1 can (7 ½ oz/213 g) pink or red salmon, drained
- Pinch of pepper
- 1 cup (250 mL) loosely packed, shredded cheddar cheese
- 2 cups (500mL) mashed potato (leftover or fresh)

Preparation

1. Mash the salmon with the bones. Put the salmon on the bottom of a small baking dish. Sprinkle it with pepper and half the shredded cheese.
2. Spread the mashed potato on top of the salmon and cheese.
3. Sprinkle the rest of the cheese on top.
4. Bake in a 350°F (180°C) oven for 30 minutes, or microwave for 8 minutes.

Tips

For a change, this dish can also be made into patties and fried in a nonstick pan.

Note: It's important to observe that every recipe in the book is structured in the same way(although there are a few exceptions, they follow a very similar pattern), so it should be easier for the LLM to come with a solid structure that should work for every dish. Since in this project we'll be working only with one book, we won't be seeing the difficulties of working with different structured data.

Methodology of the subject studied

These were the steps followed:

- Step 1: Chose the simpler recipe in the book.
- Step 2: Idealization of what would it be the best knowledge graph for that recipe.
- Step 3: Get the AI to build the knowledge graph, always with one prompt (if we did more prompts to arrange small details it would be unproductive in large scale, the objective is to come up with one general prompt for every recipe).
- Step 4: Introduction of Tasklang.
- Step 5: Compare the usage of Tasklang among different LLMs.
- Step 6: What are the differences between using Tasklang or not?
- Step 7: Experimenting with a different recipe.
- Step 8: Popularizing a knowledge graph with different recipes.
- Step 9: Using a knowledge graph to generate a recipe.

Note: in the beginning the path wasn't so clear, we did a lot of experimentation to understand which should be the workflow. At some points, I didn't really understand what was the purpose of what I was doing, but in the end, every insight that I got from the project was useful.

Initial Knowledge Graph Construction with LLMs

- Chose culinary recipes as a controlled starting point.
- Used a Seafood Burger recipe to refine prompts systematically.
- Used Mistral LLM documentation for RDF script generation.
- Two-prompt approach:

1. Extract structured data.

Input:

You are an assistant specialized in data extraction for knowledge graphs. Your task is to identify data mentioned in the text, provide a detailed description for each entity, and list its associated types so that afterwards the output you provided can be used as input to generate a knowledge graph. The entities should be proper names, concepts, procedures, or significant objects mentioned in the text. The user will supply a text that might follow the logic of a recipe. You must extract and separate the useful part of the information as ingredients, quantities, states, steps and other entities you judge as useful.

2. Convert it into RDF script.

Input:

You are an expert in knowledge graphs, ontologies, and RDF Turtle syntax. Remember that Commas are not allowed in Turtle Syntax. Your task is to map text data about recipes to the predefined classes below. You should write the output in Turtle rdf syntax that model the recipe given in the text. You can infer relationship names from the classes.

List of Classes with their attributes:

- Recipe
 - (label:str,description:str,steps:[Step],ingredients:[Ingredient],categories:[str])
- Step
 - (order:int,utensils:[Utensil],inputs:[FoodItem],outputs:[FoodItem],Action:str)
- Ingredient
 - (food_item:FoodItem,amount:[Quantity])
- Utensil
 - (label:str)
- FoodItem
 - (state:str)
- Action
 - (label:str)
- Quantity
 - (value:)
- FoodType
 - (label:str)
- Unit
 - (label:str)

Initial Knowledge Graph Construction with LLMs

- Early results showed missing details and inconsistent classifications.
- Many nodes without any meaning
- Unnecessary relationships

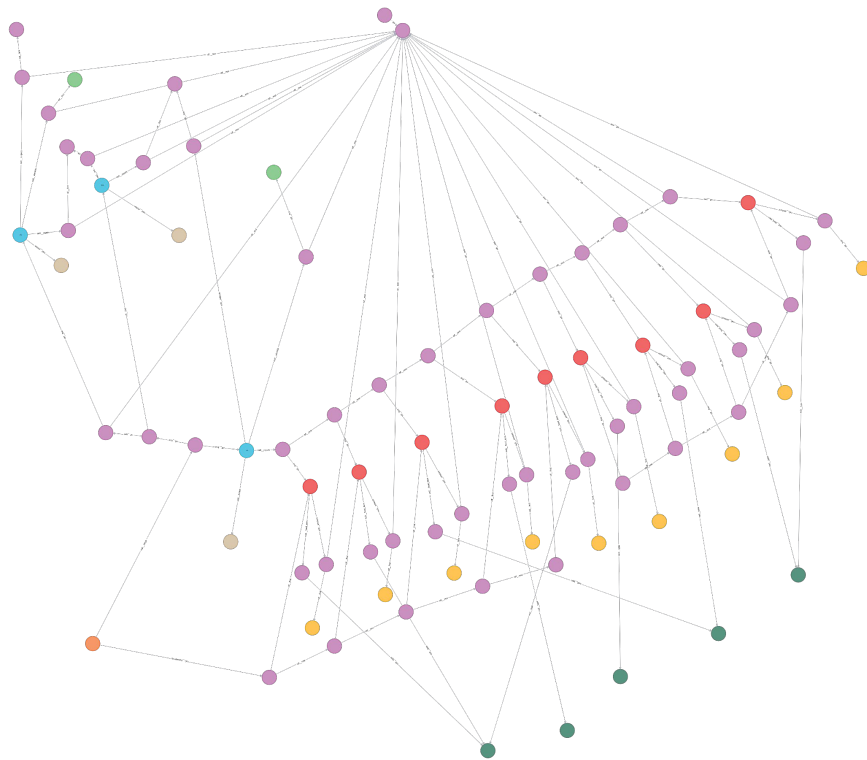
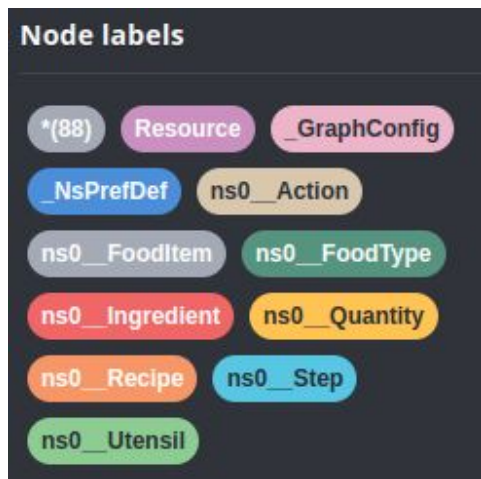


Figure 1: first knowledge graph using Mistral

Possible problem sources of first knowledge graph

At this point, some of the causes of inconsistency could be:

- LLM Limitations: Since we only used Mistral until this part, we couldn't know, but the chosen LLM could be a limiting factor.
- Prompt Design Issues: probably the main source of problems.
- Neo4j Interpretation: Some RDF scripts could be misread or poorly rendered due to semantic ambiguity.
- Lack of Structured Schema: Without a controlled ontology, entity types and relationships varied excessively.
- Learning Process: Each inconsistency revealed points for improvement, guiding the iterative refinement of prompts and modeling strategies.

Iterative Improvements

- Adjusted prompts to refine structure and retain details.
- Key challenges tackled:
 - Preserving preparation descriptions.
 - Handling alternative ingredients.
 - Improving food classification.
- Aiming for a better structured knowledge graph, I improved the first input to build a pattern before creating the graph.
- At this point, I had already identified some of the problem sources that were specifically from the recipe chosen.
- Fourth prompt.

Input:

You are an assistant specialized in data extraction for knowledge graphs. Your task is to identify data mentioned in the text, provide a detailed description for each entity, and list its associated types so that afterwards the output you provided can be used as input to generate a knowledge graph. The entities should be proper names, concepts, procedures, or significant objects mentioned in the text. If an entity is optional (serves as an alternative to another entity), you must generate a bigger entity that englobes both options, maintaining the description of each of them. The user will supply a text that might follow the logic of a recipe. You must extract and separate the useful part of the information as ingredients, quantities, states, steps, tools and other entities you judge as useful. Afterwards, your output will be used as input to an LLM to write a RDF script, so you must be objective in your classifications.

You are an expert in knowledge graphs, ontologies, and RDF Turtle syntax. Remember that commas are not allowed in Turtle syntax. Your task is to map text data about recipes to the predefined classes below.

You should write the output in Turtle RDF syntax that models the recipe given in the text. You can infer relationship names from the classes.

List of Classes with their attributes:

- **Recipe** (label: str, description: str, steps: StepStepStep, ingredients: {Ingredient}, categories: strstrstr)
- **Step** (order: int, utensils: UtensilUtensilUtensil, inputs: FoodItemFoodItemFoodItem, outputs: FoodItemFoodItemFoodItem, Action: str)
 - **Ingredient** (food_item: FoodItem, amount: Quantity)
 - **Utensil** (label: str)
 - **FoodItem** (state: str)
 - **Action** (label: str)
 - **Quantity** (value: str)
 - **FoodType** (label: str)
 - **Unit** (label: str)

Result of Iterative Improvements

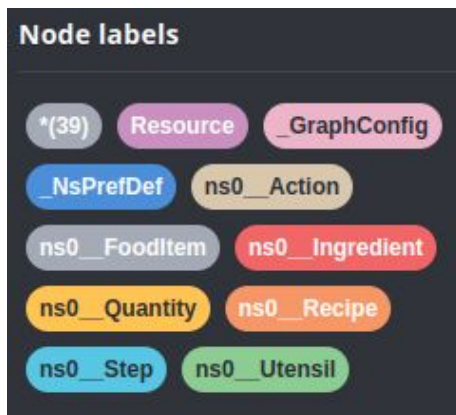
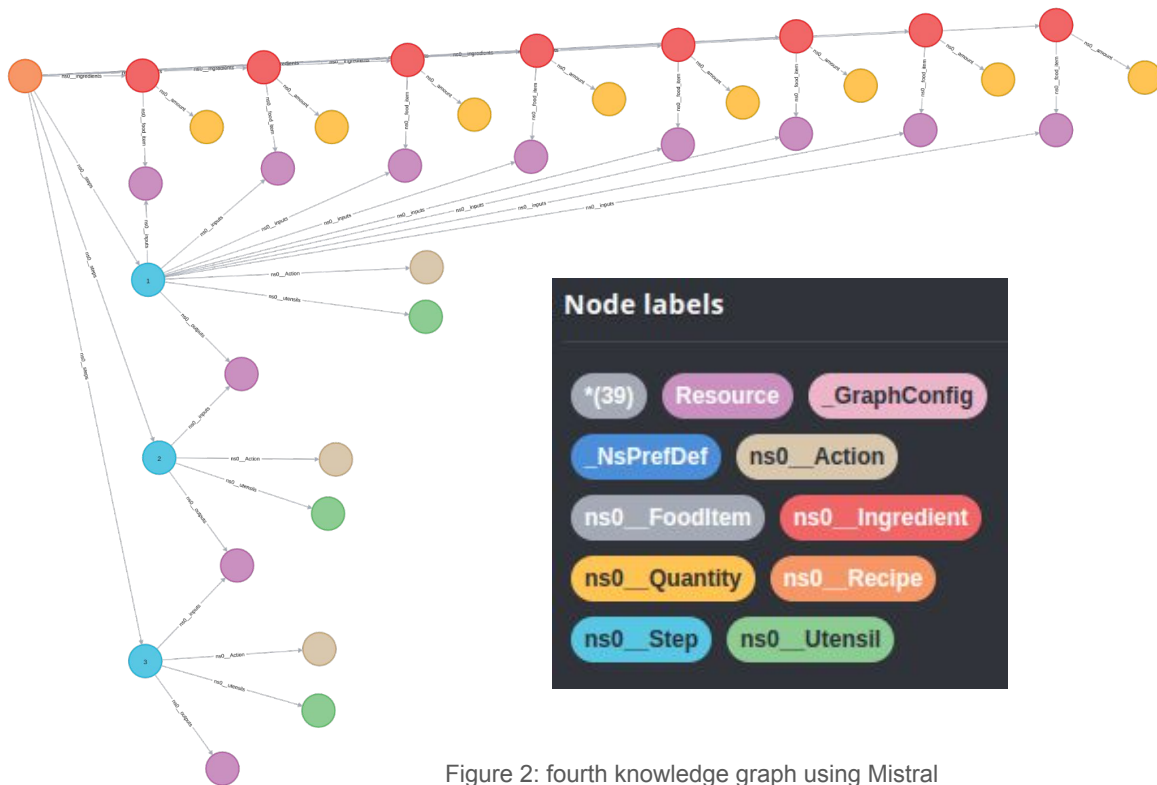


Figure 2: fourth knowledge graph using Mistral

- Better structured knowledge graph.
- Still has a logical problem due to the alternatives given in the recipe.
- Every node contain knowledge and every relationship is important.

Final Prompt Refinement

- Changed the list of class by adding a class “Alternative”.
- This change is very specific to this recipe but it's something that could appear in other dishes.
- Sixth prompt

Input:

You are an assistant specialized in data extraction for knowledge graphs. Your task is to identify data mentioned in the text, provide a detailed description for each entity, and list its associated types so that afterwards the output you provided can be used as input to generate a knowledge graph. The entities should be proper names, concepts, procedures, or significant objects mentioned in the text. If an entity is optional (serves as an alternative to another entity), you must generate a bigger entity that englobes both options, maintaining the description of each of them. The user will supply a text that might follow the logic of a recipe. You must extract and separate the useful part of the information as ingredients, quantities, states, steps, tools and other entities you judge as useful. Afterwards, your output will be used as input to an LLM to write a RDF script, so you must be objective in your classifications.

You are an expert in knowledge graphs, ontologies, and RDF Turtle syntax. Remember that commas are not allowed in Turtle syntax. Your task is to map text data about recipes to the predefined classes below. You should write the output in Turtle RDF syntax that models the recipe given in the text. You can infer relationship names from the classes. Separate options in a logical way.

List of Classes with their attributes:

- Recipe (label: str, description: str, steps: StepStepStep, ingredients: {Ingredient}, categories: strstrstr)
- Step (order: int, utensils: UtensilUtensilUtensil, inputs: FoodItemFoodItemFoodItem, outputs: FoodItemFoodItemFoodItem, Action: str)
 - Ingredient (alternatives: {Alternative})
 - Alternative (food_item: FoodItem, amount: Quantity)
 - Utensil (label: str)
 - FoodItem (state: str)
 - Action (label: str, description: str)
 - Quantity (value: str)

Final Prompt Refinement

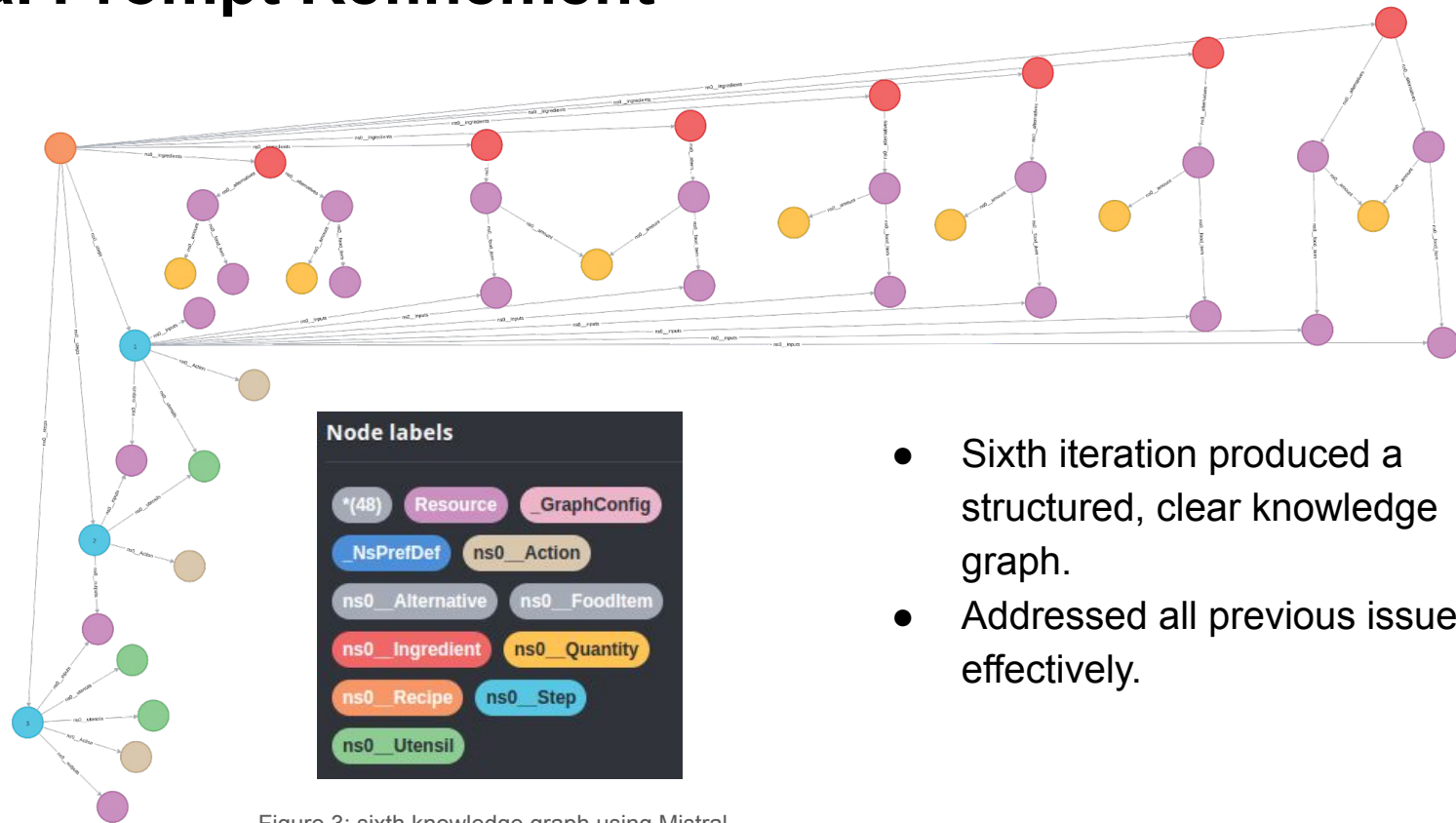


Figure 3: sixth knowledge graph using Mistral

- Sixth iteration produced a structured, clear knowledge graph.
- Addressed all previous issues effectively.

Final Conclusion about the Prompt Refinement

- Refined prompts significantly improved the structure and coherence of the generated knowledge graphs.
- Most major issues were resolved, including classification, detail preservation, and relation accuracy.
- However, prompts alone were not sufficient to ensure full consistency and standardization across outputs.
- The process revealed the need for a more robust, structured approach to guide the LLM.
- This insight led to the introduction of TaskLang, a templated method designed to enforce schema and reduce ambiguity.

Note: While prompt engineering brought substantial progress, it also showed the limitations of relying solely on natural language. Natural language is inherently ambiguous and context-dependent, which makes it difficult to consistently produce structured, machine-readable outputs, especially when the task requires precision, hierarchy, and semantic relationships, as is the case in knowledge graphs.

Tasklang

```
# TaskLang : Un Modèle Universel de Description de Tâches (Inspiré de AROMI)

## 1. Structure Fondamentale

### PROBLEM <Problem_Name>
* **@RDFClass** : 'tasklang:Problem'
* **@URI** : '<http://example.org/problems/ProblemName>'

**DESCRIPTION** : "<Description détaillée du problème à résoudre>"
**GOAL** : "<Objectif principal ou résultat attendu>"

**INPUTS** :
- resources : [
  * **@RDFClass** : 'tasklang:Resource'
  * **@Nom_Ressource_1** : "<Description détaillée> (<quantité>/mesure), <qualité>"
  * **@Nom_Ressource_2** : "<Description détaillée> (<quantité>/mesure), <qualité>"
]
- design : "<Description ou schéma du processus/approche>"
- context : {
  * **@RDFClass** : 'tasklang:Context'
  domain : "<Domaine d'application>",
  constraints : "<Contraintes spécifiques>",
  reference : "<Référence ou source principale>",
  parameters : [
    {
      * **@RDFClass** : 'tasklang:Parameter'
      name : "<Nom Paramètre_1>",
      value : "<Valeur>",
      unit : "<Unité>",
      valid_range : "<Intervalle valide>"
    },
    {
      name : "<Nom Paramètre_2>",
      value : "<Valeur>",
      unit : "<Unité>",
      valid_range : "<Intervalle valide>"
    }
  ]
}

**QUALITY_CRITERIA** :
- "<Critère_1> : <Description du critère et méthode d'évaluation>"
- "<Critère_2> : <Description du critère et méthode d'évaluation>"

**OUTPUTS** :
- finalProduct : "<Description du résultat final attendu>"
- secondaryOutputs : [
  "<Output_Secondaire_1> : <Description>",
  "<Output_Secondaire_2> : <Description>"
]

### METHOD <Method_Name>
* **@RDFClass** : 'tasklang:Method'
* **@URI** : '<http://example.org/methods/MethodName>'
* **@DomainReference** : '<http://domain-specific-ontology.org/concept/Refer'

**DOMAIN** : "<Domaine d'application>"
**DESCRIPTION** : "<Description détaillée de la méthode>"
**REFERENCE** : "<Sources, références, ou origine de la méthode>"

**PREREQUISITES** :
- "<Pré-requis_1>" [**@Importance** : "<Critique/Important/Recommandé>"]
- "<Pré-requis_2>" [**@Importance** : "<Critique/Important/Recommandé>"]

**STEPS** :
1. **TASK** "<Nom Tâche_1>"
  * **@RDFClass** : 'tasklang:Task'
  * **@DomainTechnique** : '<http://domain-specific-ontology.org/techniques'

  **INPUTS** : [<liste des entrées requises>]
  **ACTION** : "<Description détaillée de l'action à effectuer>"
  **NOTES** : "<Informations complémentaires, astuces, conseils>"
  **DURATION** : "<Estimation du temps requis>"
  **OUTPUT** : "<Description du résultat de cette tâche>"

  **VALIDATION_RULES** : [
    * **@RDFClass** : 'tasklang:ValidationRule'
    "<Règle Validation_1> : <Description de la règle>"
  ]

  **PARAMETERS** : [
    {
      name : "<Nom Paramètre>",
      value : "<Valeur>",
      measurement : "<Méthode de mesure ou vérification>"
    }
  ]

2. **TASK** "<Nom Tâche_2>"
  * **@RDFClass** : 'tasklang:Task'

  **ACTION** : "<Description détaillée de l'action à effectuer>"
  **NOTES** : "<Informations complémentaires>"
  **DURATION** : "<Estimation du temps requis>"
  **OUTPUT** : "<Description du résultat de cette tâche>"

3. **DECISION** "<Nom Décision>"
  * **@RDFClass** : 'tasklang:Decision'

  **CONDITION** : "<Condition à évaluer>"
  **OPTIONS** :
  - **@OPTION** "<Nom Option_1>":
    **WHEN** : "<Condition spécifique>"
    **THEN** : "Aller à TASK '<Nom Tâche Cible_1>'"
    **REASON** : "<Justification du choix>"
  - **@OPTION** "<Nom Option_2>":
    **WHEN** : "<Condition spécifique>"
    **THEN** : "Aller à TASK '<Nom Tâche Cible_2>'"
    **REASON** : "<Justification du choix>"

4. **LOOP** "<Nom Boucle>"
  * **@RDFClass** : 'tasklang:Loop'

  **CONDITION** : "<Condition de sortie>"
  **BLOCK** :
  a. **TASK** "<Nom Sous-Tâche_1>"
    **ACTION** : "<Description détaillée de l'action à effectuer>"
    **DURATION** : "<Estimation du temps requis>"
    **OUTPUT** : "<Description du résultat>"
  b. **TASK** "<Nom Sous-Tâche_2>"
    **ACTION** : "<Description détaillée de l'action à effectuer>"
    **NOTES** : "<Informations complémentaires>"
    **DURATION** : "<Estimation du temps requis>"
    **OUTPUT** : "<Description du résultat>"
  c. **MEASUREMENT** "<Nom Mesure>"
    **ACTION** : "<Comment effectuer la mesure>"
    **CONDITION** :
    - **EXPRESSION** : "<Expression conditionnelle basée sur la mesure>"
    - **CONSEQUENCE** : "<Action à entreprendre si la condition est vraie>"
    **OUTPUT** : "<Résultat de la mesure>"

5. **PARALLEL** "<Nom Tâches Parallèles>"
  * **@RDFClass** : 'tasklang:ParallelTasks'

  **DESCRIPTION** : "<Description des tâches à exécuter en parallèle>"
  **TASKS** :
  - **TASK** "<Nom Tâche Parallèle_1>"
    **ACTION** : "<Description détaillée>"
    **DURATION** : "<Estimation du temps requis>"
  - **TASK** "<Nom Tâche Parallèle_2>"
    **ACTION** : "<Description détaillée>"
    **DURATION** : "<Estimation du temps requis>"

  **SYNCHRONIZATION** : "<Condition ou point de synchronisation>"

6. **TASK** "<Nom Tâche Finale>"
  * **@RDFClass** : 'tasklang:Task'

  **ACTION** : "<Description détaillée de l'action à effectuer>"
  **NOTES** : "<Informations complémentaires>"
  **DURATION** : "<Estimation du temps requis>"
  **OUTPUT** : "<Description du résultat final>"

  **RISKS** :
  - "<Risque_1> : <Description du risque>" [**@Severity** : "<Haute/Moyenne/Basse>"]
  - "<Risque_2> : <Description du risque>" [**@Severity** : "<Haute/Moyenne/Basse>"]

  **VALIDATION_RULES** :
  * **@RDFClass** : 'tasklang:ValidationRule'
  * "SI <condition> ALORS <conséquence>"
  * "SI <condition> ALORS <conséquence>"
```

Tasklang

- Introduced to improve structure and consistency.
- Initial misunderstanding: Evaluated LLMs instead of TaskLang's effectiveness.
- Prompt:

You are an expert in knowledge graphs, ontologies, and RDF Turtle syntax. Remember that commas are not allowed in Turtle syntax. Your task is to map text data about recipes to the predefined classes below. You should write the output in Turtle RDF syntax that models the recipe given in the text. You can infer relationship names from the classes. Separate options in a logical way. Structure all the existing knowledge according to the Tasklang attached in this message

+ RECIPE

+ TASKLANG

Tasklang

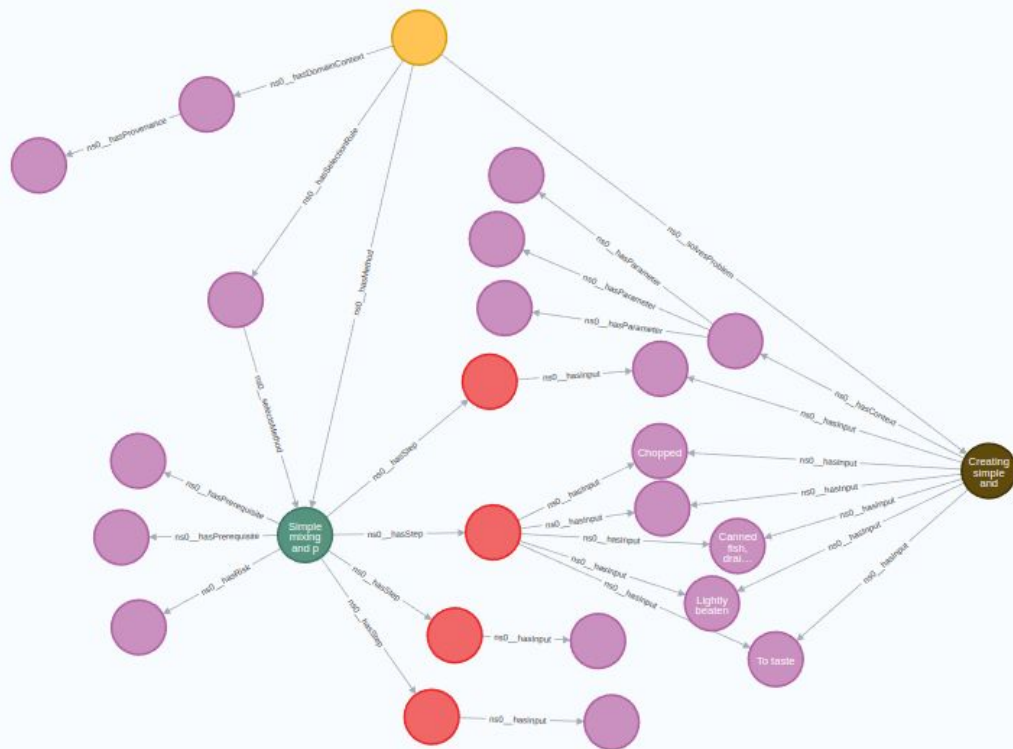


Figure 4: first graph made by Claude using Tasklang

Node labels

*(33)	Resource	_GraphConfig
_NsPrefDef	ns0__Context	
ns0__DomainContext		
ns0__Method	ns0__Parameter	
ns0__Prerequisite	ns0__Problem	
ns0__Provenance	ns0__Resource	
ns0__Risk	ns0__SelectionRule	
ns0__Strategy	ns0__Task	

Tasklang

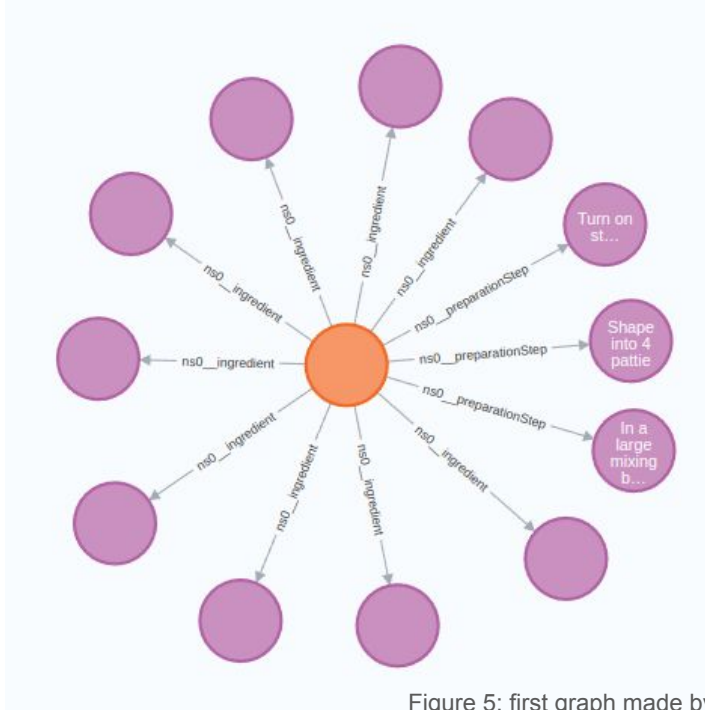


Figure 5: first graph made by Mistral using Tasklang



Tasklang

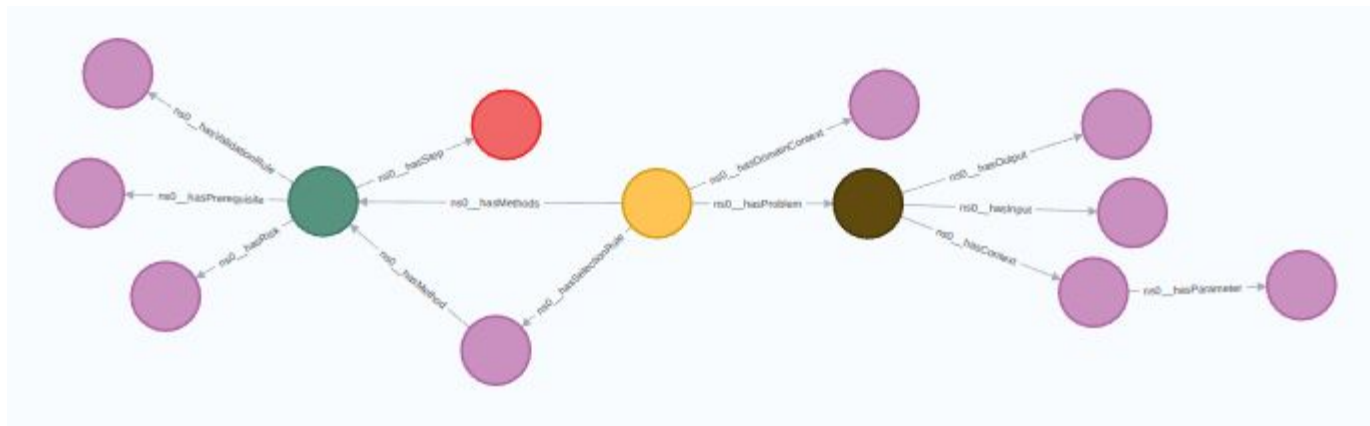


Figure 6: first graph made by Perplexity using Tasklang



Tasklang

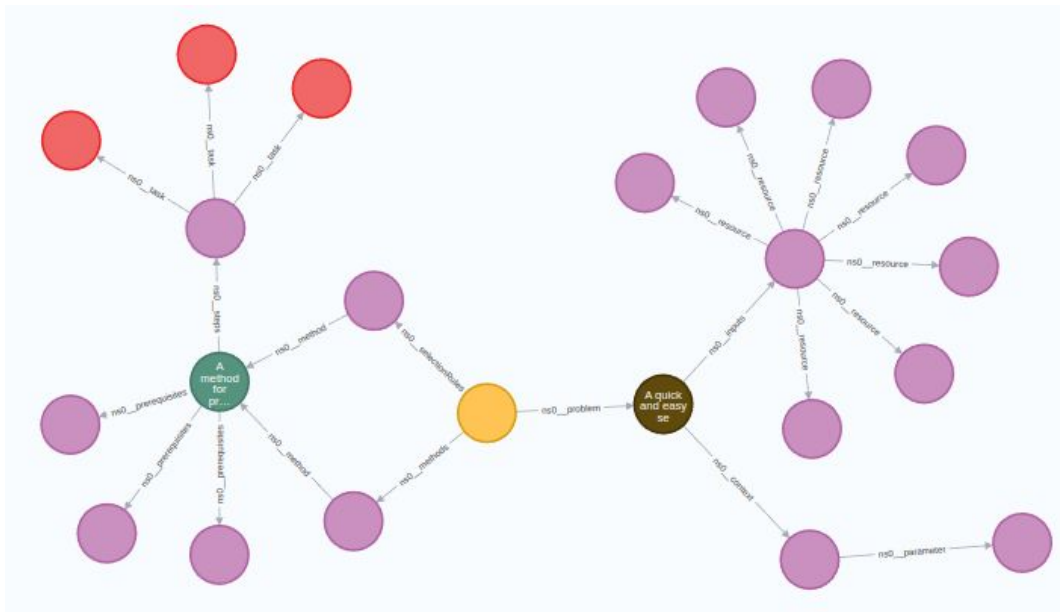


Figure 7: first graph made by ChatGPT using Tasklang

Interpretation of the knowledge graphs

- After initial misunderstanding, refocused on measuring improvements TaskLang brings to knowledge graphs.
- Mistral, the AI we used until here, didn't follow the Tasklang as the others.
- ChatGPT, Claude and Perplexity both followed the structure proposed, specially Claude and ChatGPT, as Perplexity built a simpler knowledge graph.
- Among ChatGPT and Claude, we chose to continue with Claude because of the completeness.
- Elaboration of a new prompt to use combined with Tasklang.
- Few steps back
 - Reevaluated Claude's performance without Tasklang
 - Same prompt as used in Mistral
 - Assure that the difference is not caused by the LLM selection

What are the differences between using Tasklang or not?

- Compared knowledge graphs with and without TaskLang but built with the same LLM to evaluate structure and consistency.
- Focused on key criteria:
 - Graph size and clarity
 - Presence of unnecessary nodes and relationships
 - Structural coherence and readability
- To measure consistency, I developed three different graphs using Tasklang. This way, we are able to identify whether Tasklang provides the AI a solid pattern to follow or not.

Evaluation Without Tasklang

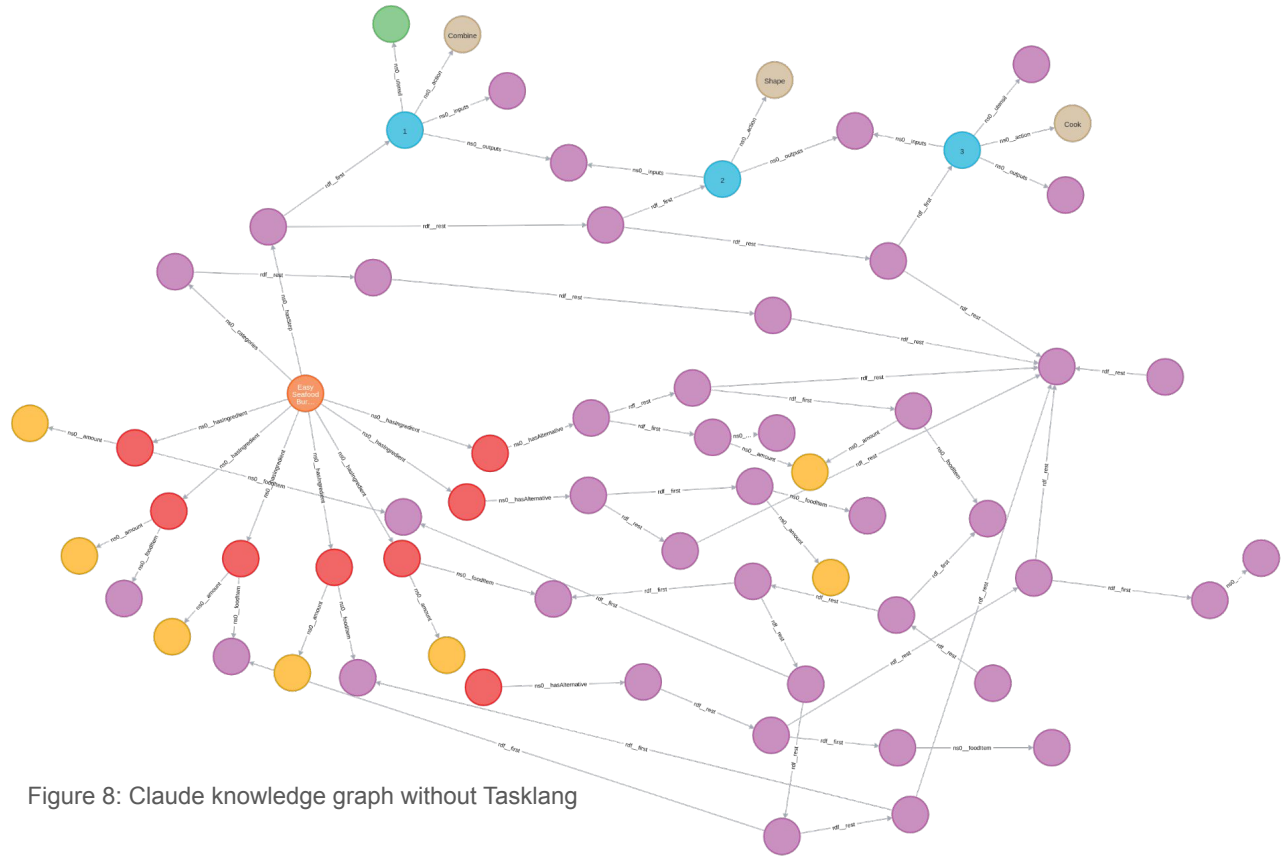


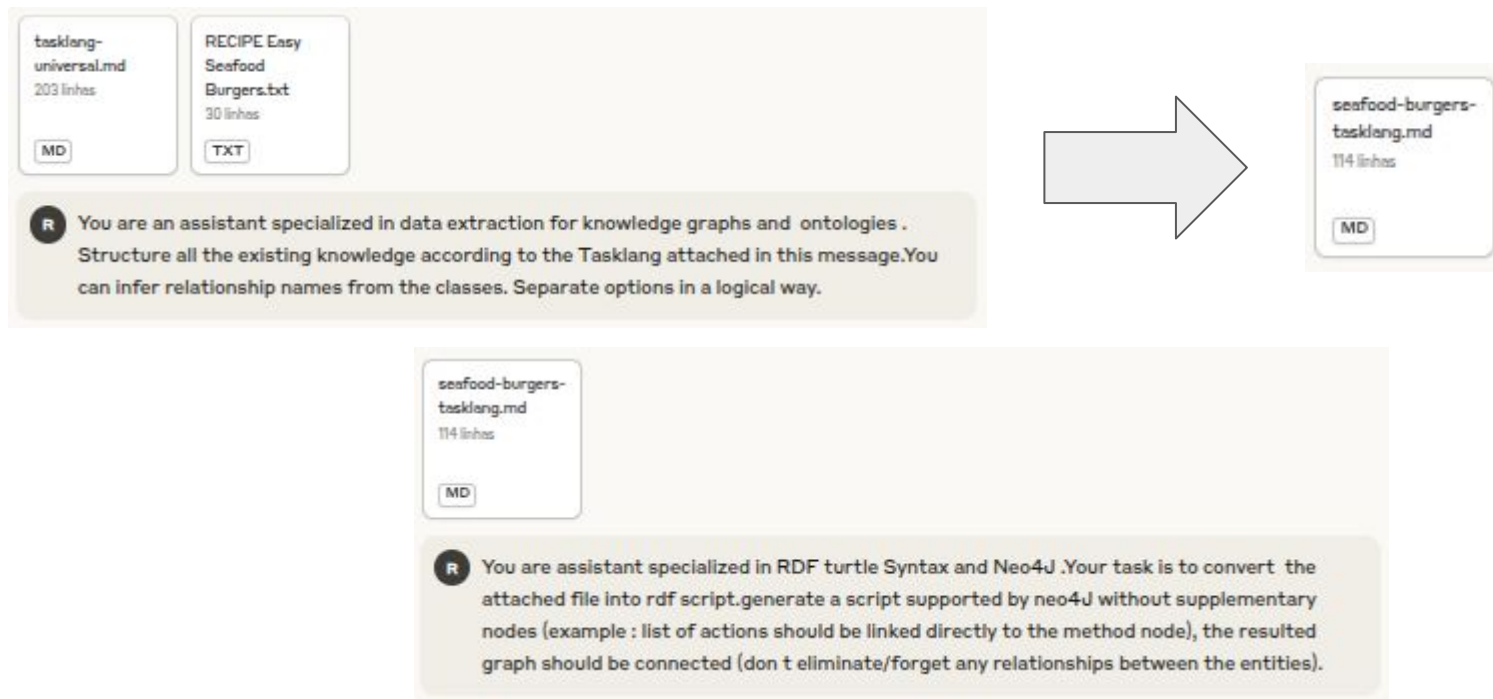
Figure 8: Claude knowledge graph without Tasklang



- Lack of structure and standardization.
- Unclear relationships between ingredients and steps.
- Graphs were large and inconsistent.

Evaluation With Tasklang

- First prompt



Evaluation With Tasklang

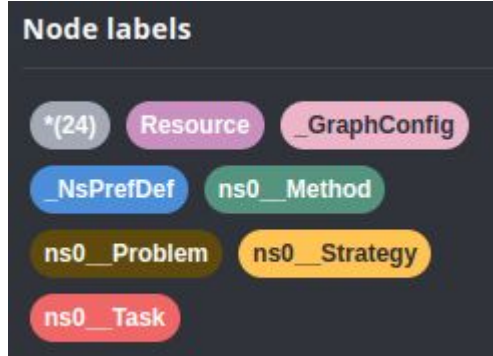
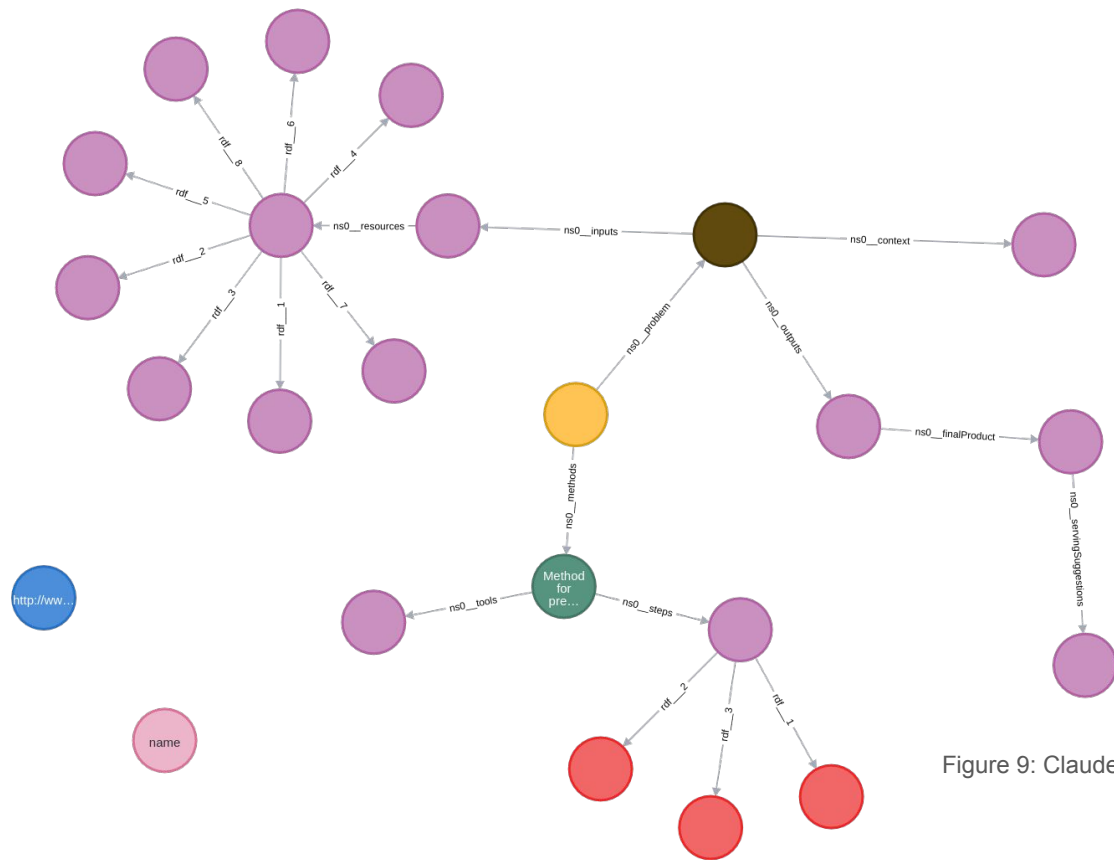


Figure 9: Claude knowledge graph with Tasklang first prompt

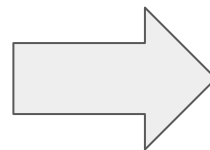
Evaluation With Tasklang

- Second prompt

tasklang-universal.md
203 linhas
MD

RECIPE Easy
Seafood
Burgers.txt
30 linhas
TXT

R You are an assistant specialized in data extraction for knowledge graphs and ontologies .
Structure all the existing knowledge according to the Tasklang attached in this message.You
can infer relationship names from the classes. Separate options in a logical way.



seafood-burgers-
tasklang.md
114 linhas
MD

seafood-burgers-
tasklang.md
114 linhas
MD

R Translate the following user file to an RDF graph using the RDF, RDFS, and OWL ontologies
formatted as TTL.Use the prefix ex: with IRI <http://example.com/> for any created entities.

Evaluation With Tasklang

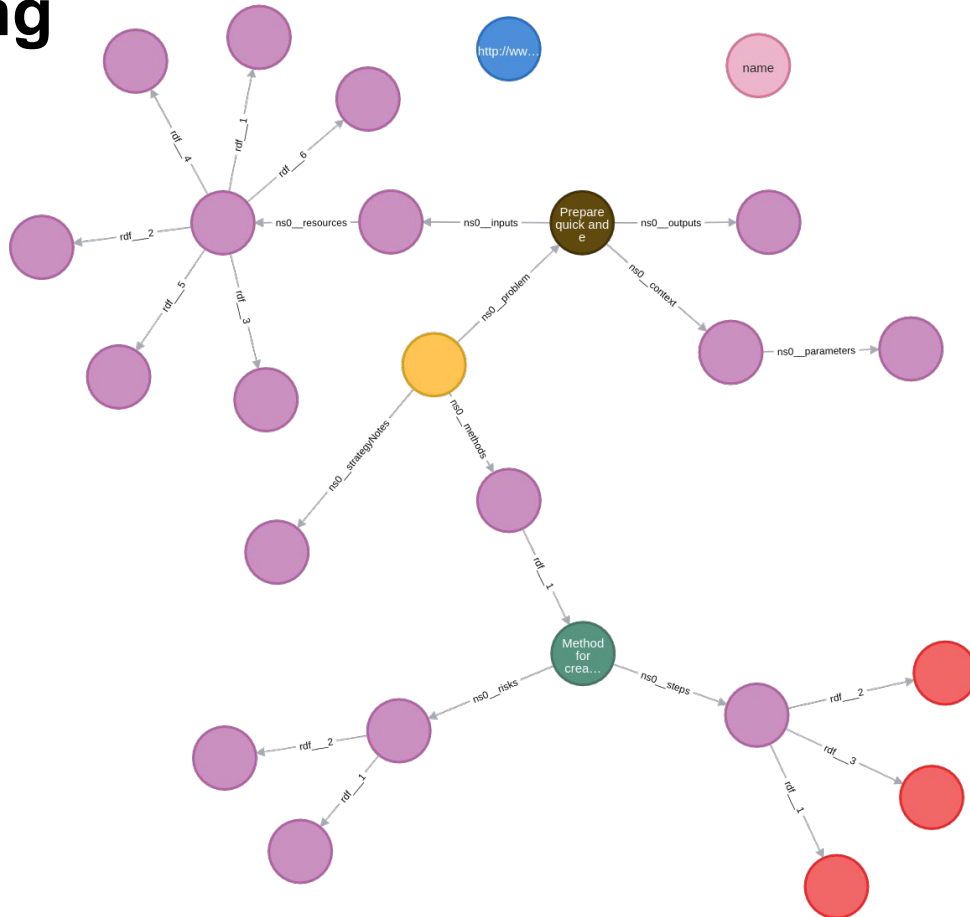
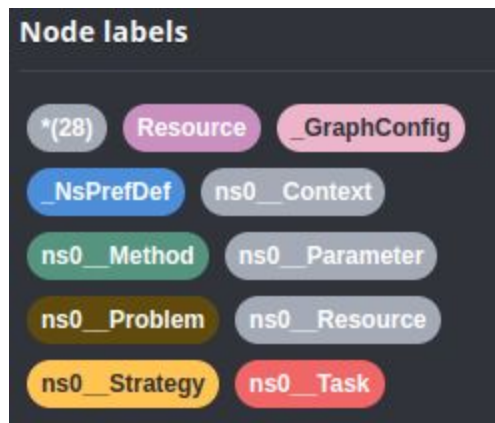


Figure 10: Claude knowledge graph with Tasklang second prompt

Evaluation With Tasklang

- Third prompt

RECIPE Easy
Seafood
Burgers.txt
30 lines

TXT

tasklang-
universal.md
203 lines

MD

R You will receive a recipe and a TaskLang template that defines the entities and relationships expected in an RDF Turtle script. Your task is to transform the recipe into a valid RDF Turtle script that conforms to the provided TaskLang schema.

Requirements:

- Strict Schema Compliance:** The generated RDF script must strictly adhere to the structure defined in TaskLang. Ensure that all required entities and relationships are present.
- Correct Turtle Syntax:** The output must be a syntactically valid Turtle RDF script. Validate that prefixes, URIs, literals, and relationships are properly formatted.
- Meaningful Entity Mapping:** Extract relevant information from the recipe and map it to the correct entities in TaskLang. If an entity is missing, infer it logically where possible.
- Clear Relationships:** Define relationships clearly, ensuring that each entity connects properly within the knowledge graph structure.
- Consistent Formatting:** Maintain readability by organizing the RDF script properly, using indentation and meaningful labels for URIs.

Input Structure: All the data is attached in this chat:

- A **recipe**, formatted as plain text.
- A **TaskLang template**, which specifies the expected structure for the RDF representation.

Output Structure: Your output must be a valid Turtle RDF script that fully represents the recipe according to the TaskLang schema.

Evaluation With Tasklang

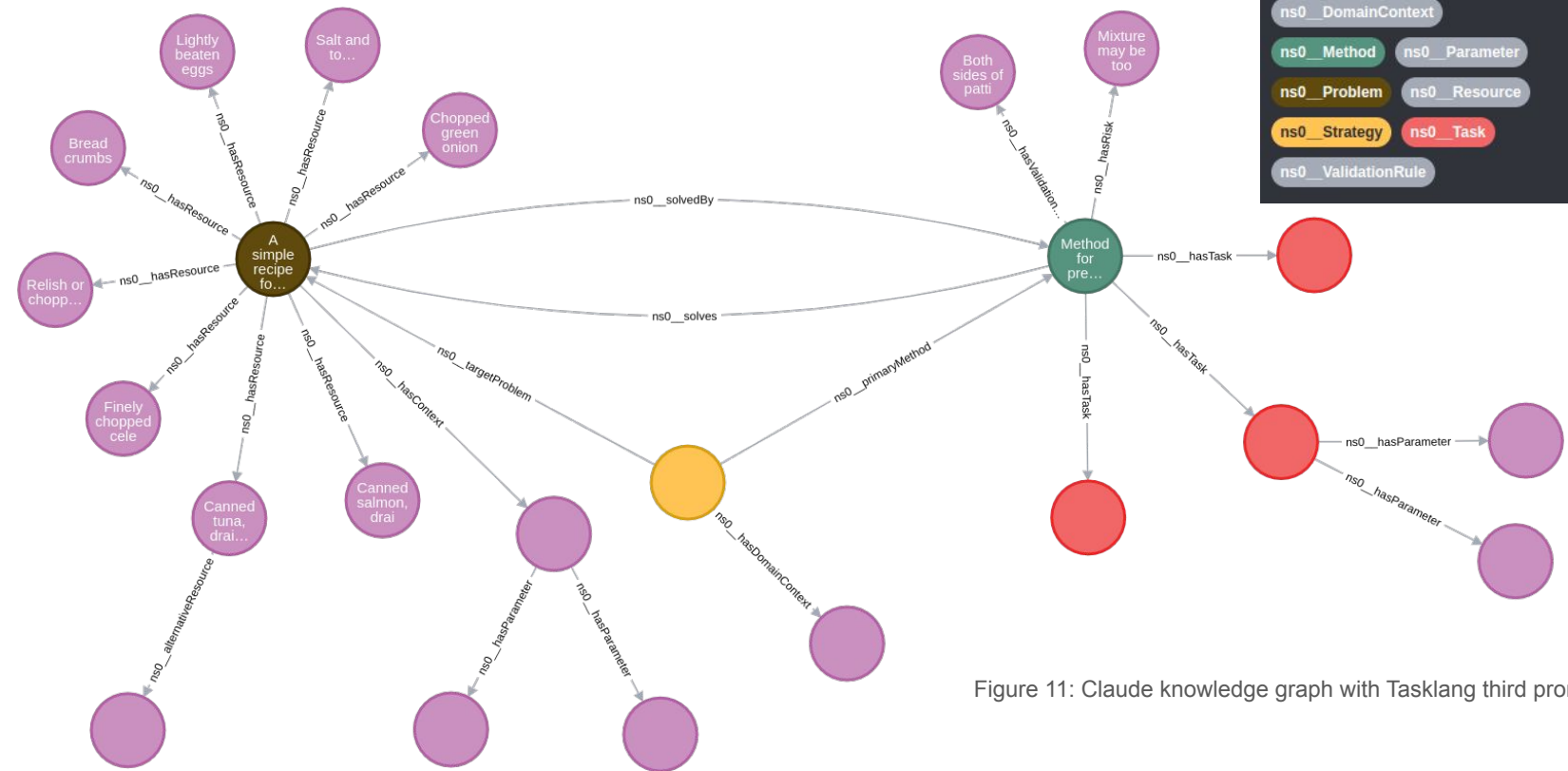


Figure 11: Claude knowledge graph with Tasklang third prompt

Evaluation With Tasklang

- Enforced structured schema and standardized entities.
- Improved clarity and queryability.
- Needed refinements in relationship labels and ingredient mappings.
- Besides the changes in the prompt, is evident the similarity between each knowledge graph

Experimenting with a different recipe.

- Following the objective of the whole project that is to build a gigantic knowledge graph with many different recipes, I decided to test what I already had for another recipe.
- First try of the same prompt with a Salmon Potato Dish Recipe.
- It looked good, although different from the pattern we got used to with the other recipe.

Note: as I was working with a new recipe, I was in doubt if the difference seen between the graphs of different recipes was caused because of the LLM or because of the recipe. So I decided to test again.

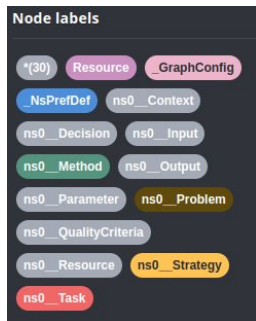


Figure 12: Claude Knowledge Graph of Salmon Potato Dish

Experimenting with a different recipe.

Note: when interacting with LLMs, especially in multi-turn conversations, the model tends to adapt to the conversation's context and establish internal patterns based on previous messages. So I tried the new dish in the same chat as the original Salmon Potato Recipe.

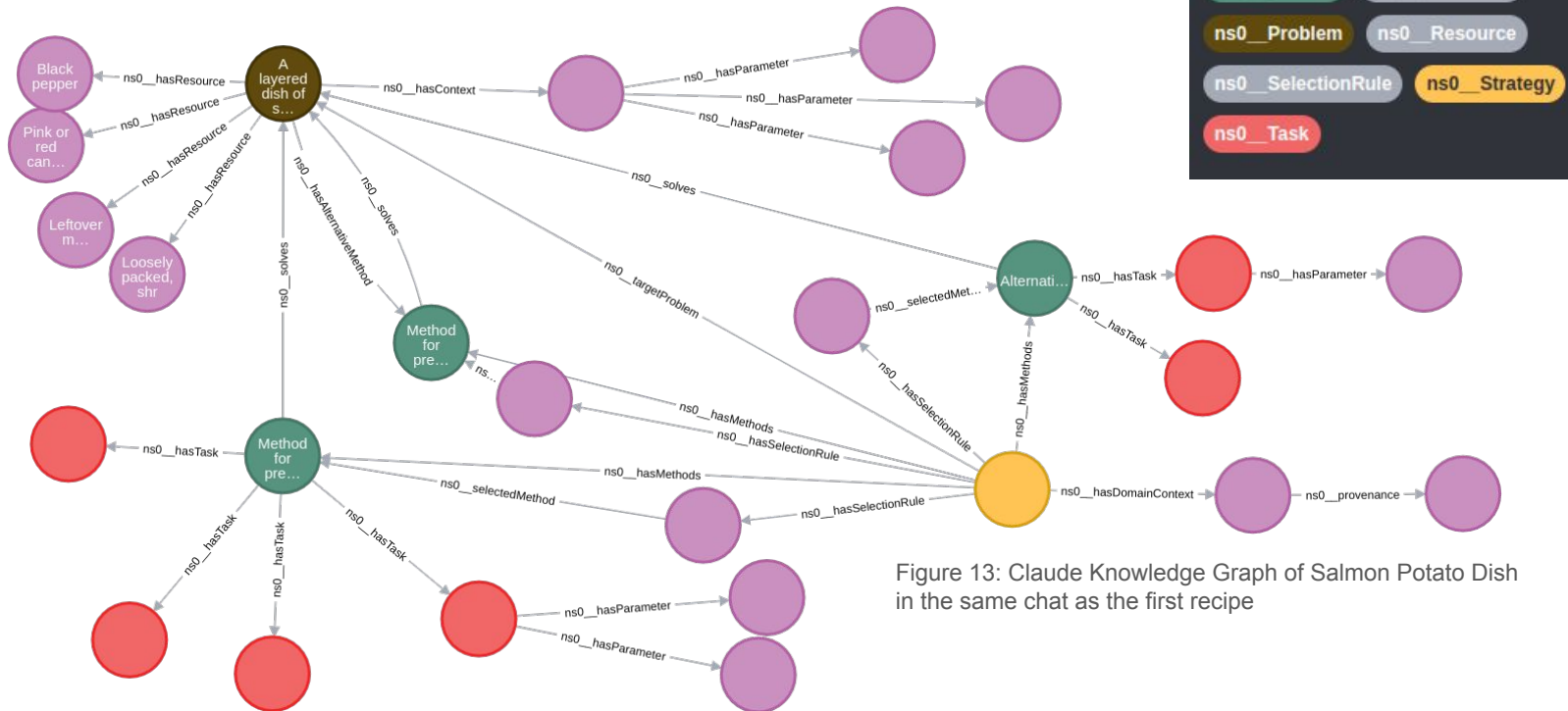


Figure 13: Claude Knowledge Graph of Salmon Potato Dish in the same chat as the first recipe

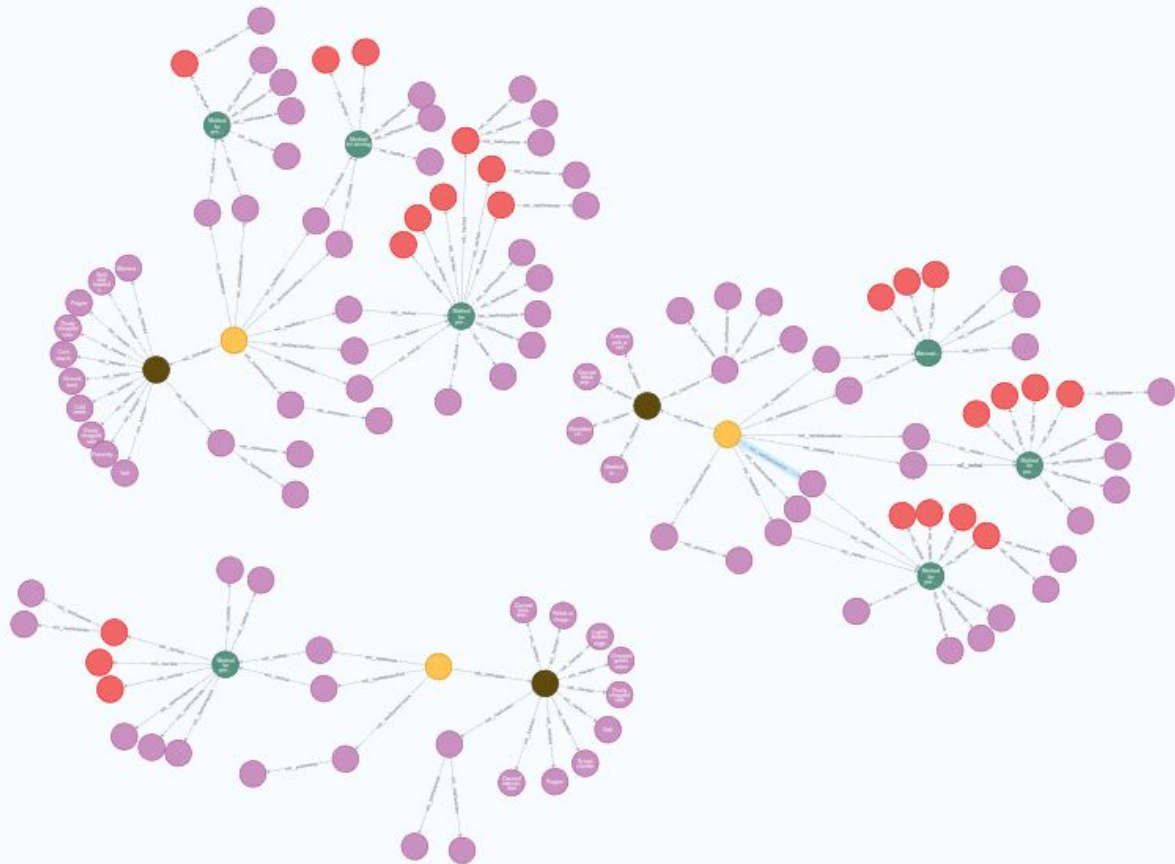
Conclusion evaluation with tasklang

- TaskLang significantly improved structure, clarity, and consistency in knowledge graphs.
- All three graphs generated using TaskLang shared a notable similarity, indicating reproducibility.
- Even if the recipe changes, Tasklang still provides a solid pattern.
- Claude showed strong adherence to the template, producing graphs with minimal noise.
- Structural patterns remained intact across different executions, confirming that TaskLang enforces a usable schema.
- TaskLang is not just a format; it's a reliable strategy to guide LLMs when generating complex structured outputs.

Recipes Knowledge Graph

- First attempt at combining multiple recipes in a single graph.
- Recipes included were not constrained by a strict ontology.
- The goal was to test how TaskLang and the LLM handled aggregation of structured data.
- Results showed potential, but highlighted the need for better entity linking and recipe separation.

Recipes Knowledge Graph



Node labels

*(143) Resource _GraphConfig
_NsPrefDef ns0__Context
ns0__DomainContext
ns0__Method ns0__Parameter
ns0__Problem ns0__Resource
ns0__Rule ns0__Strategy
ns0__Task

First attempt of gathering three different recipes in one graph, without constraint

Future Directions

1. Refining prompts within TaskLang for even better accuracy.
2. Testing a "reverse" process: Using a knowledge graph to generate a recipe.
3. Enhancing entities and relationship modeling for better graph connections.
4. Expanding knowledge graphs to other domains (medical, DIY, industrial).
5. Integrating external datasets (nutrition, professional guidelines).
6. Developing an interactive recipe search engine with natural language.

Among the items above, I was able to explore the item 2 and 6.

Using a knowledge graph to generate a recipe

What's the idea?

- RDF from an existing graph was provided as input to the LLM.
- Prompt example:
 - “Given this RDF knowledge graph, generate a complete and coherent recipe including ingredients and steps.”
- LLM interpreted RDF data and generated a recipe.
- Recipe generated by the LLM is compared to the original recipe.

What we were doing:

RECIPE
Easy Seafood Burgers
Ready in **10 minutes**
Serves **4 people**

Ingredients

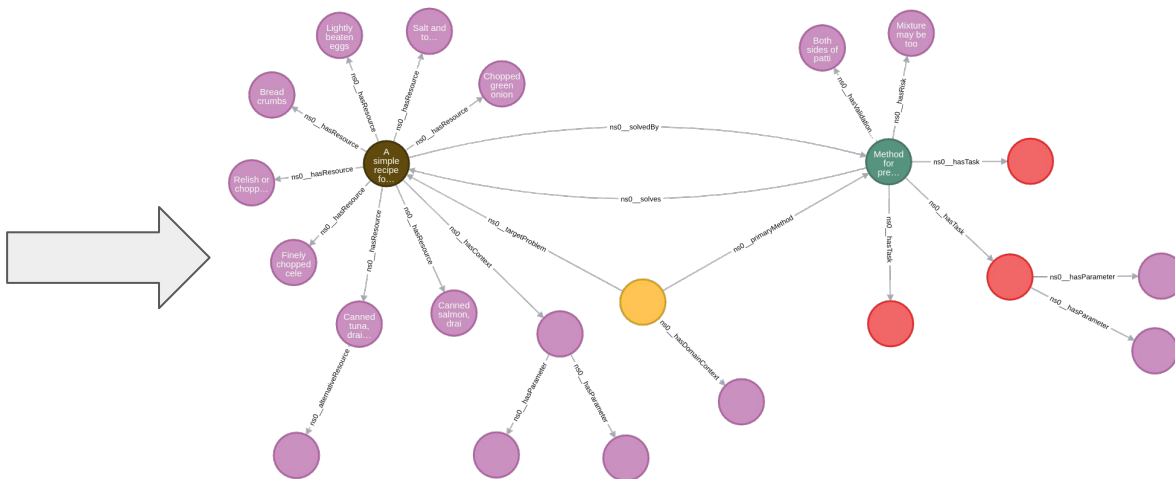
- 1 can (170g) tuna or can (7 ½ oz/213g) salmon, drained (if using salmon, remove skin and large bones)
- ½ cup (60mL) celery, finely chopped
- 2 Tbsp (30mL) relish or chopped pickles
- 2 eggs, lightly beaten
- ½ cup (125mL) bread crumbs
- 2 green onions, chopped
- Salt and pepper to taste

Preparation

1. In a large mixing bowl, combine all ingredients.
2. Shape into 4 patties.
3. Turn on the stove to medium-high heat. In a non-stick pan, cook patties until both sides are golden brown - about 3 minutes per side.

Tips

This recipe makes 4 small patties and tastes just as delicious with canned salmon.
Serve on whole wheat toast, bannock, or a bun.



Using a knowledge graph to generate a recipe

What we did:



I am sending you a recipe represented as a knowledge graph in RDF script attached to this message and I want you to return the recipe with clear instructions that can be understood by a human user. Gather all the information of the graph.

Easy Seafood Burgers

Ready in 10 minutes | Serves 4

Ingredients

- 170g (6oz) canned tuna, drained OR 213g (7½oz) canned salmon, drained with skin and large bones removed
- ¼ cup (60mL) finely chopped celery
- 2 Tbsp (30mL) relish or chopped pickles
- 2 lightly beaten eggs
- ½ cup (125mL) bread crumbs
- 2 chopped green onions
- Salt and pepper to taste

Instructions

1. In a large mixing bowl, combine all ingredients (fish, celery, relish, eggs, bread crumbs, green onions, salt, and pepper).
2. Shape the mixture into 4 patties.
3. Heat a non-stick pan over medium-high heat.
4. Cook patties until golden brown on both sides, about 3 minutes per side (6 minutes total).

Serving Suggestion

Serve on whole wheat toast, bannock, or a bun.

Tips

- If the mixture is too wet and the patties don't hold together well, add more bread crumbs.
- This recipe works equally well with either canned tuna or salmon.
- This is a quick and easy recipe suitable for beginners.

Enjoy your delicious seafood burgers!

Figure 15: Knowledge Graph generated by Claude with Tasklang

Using a knowledge graph to generate a recipe

- Comparing them both, we see that no information was lost during the conversion process:

RECIPE

Easy Seafood Burgers

Ready in **10 minutes**

Serves **4 people**

Ingredients

- 1 can (170g) tuna or can (7 ½ oz/213g) salmon, drained (if using salmon, remove skin and large bones)
- ¼ cup (60mL) celery, finely chopped
- 2 Tbsp (30mL) relish or chopped pickles
- 2 eggs, lightly beaten
- ½ cup (125mL) bread crumbs
- 2 green onions, chopped
- Salt and pepper to taste

Tips

This recipe makes 4 small patties and tastes just as delicious with canned salmon.

Serve on whole wheat toast, bannock, or a bun.

Preparation

- In a large mixing bowl, combine all ingredients.
- Shape into 4 patties.
- Turn on the stove to medium-high heat. In a non-stick pan, cook patties until both sides are golden brown - about 3 minutes per side.

Original recipe

Easy Seafood Burgers

Ready in 10 minutes | Serves 4

Ingredients

- 170g (6oz) canned tuna, drained OR 213g (7½oz) canned salmon, drained with skin and large bones removed
- ¼ cup (60mL) finely chopped celery
- 2 Tbsp (30mL) relish or chopped pickles
- 2 lightly beaten eggs
- ½ cup (125mL) bread crumbs
- 2 chopped green onions
- Salt and pepper to taste

Instructions

- In a large mixing bowl, combine all ingredients (fish, celery, relish, eggs, bread crumbs, green onions, salt, and pepper).
- Shape the mixture into 4 patties.
- Heat a non-stick pan over medium-high heat.
- Cook patties until golden brown on both sides, about 3 minutes per side (6 minutes total).

Serving Suggestion

Serve on whole wheat toast, bannock, or a bun.

Tips

- If the mixture is too wet and the patties don't hold together well, add more bread crumbs.
- This recipe works equally well with either canned tuna or salmon.
- This is a quick and easy recipe suitable for beginners.

Enjoy your delicious seafood burger!

LLM recipe

Developing an interactive recipe search engine with natural language

- Retrieval-Augmented Generation (RAG)
 - RAG combines the generative power of LLMs with structured retrieval from a knowledge source, as a graph
 - In this project, RAG is used to generate answers based on actual graph data, reducing hallucinations and improving factual accuracy.
- How It Works
 - The LLM receives a query.
 - It retrieves relevant triples or nodes from the graph (via Cypher queries or other structured access).
 - The LLM uses this structured input to generate contextual, accurate responses.
- Enables explainability and grounding in data.

Developing an interactive recipe search engine with natural language



- Neo4j Graph Builder
 - This tool offers a natural language interface powered by an embedded LLM.
 - You can ask complex questions such as:
 - “Which recipes use salmon and no dairy?”
 - “What are the alternative ingredients for lemon?”
 - The embedded LLM interprets the question and will return an answer
 - No need to know cypher queries to select knowledge, you can use natural language.

Developing an interactive recipe search engine with natural language



The steps for preparing the "Easy Seafood Burgers" are:
Combine Ingredients: Mix all ingredients in a large mixing bowl. This step takes approximately 2 minutes and results in a mixed seafood burger mixture.
Shape Patties: Form the mixture into 4 equal patties. This step takes about 1 minute and produces 4 raw seafood burger patties.



6/2/2025, 3:16:46 PM

[Details](#)  



Yes, the final step in the "Easy Seafood Burgers" recipe involves cooking the patties. You need to cook the patties in a non-stick pan at medium-high heat until they are golden brown. This step takes approximately 6 minutes, with 3 minutes per side.

6/2/2025, 3:17:20 PM

[Details](#)  

what are the steps

6/2/2025 3:16:46 PM



i have to cook something?

6/2/2025 3:17:20 PM



Material of the internship

- All project materials are available in the [GitHub](#) repository named “Knowledge-graphs”.
- Detailed information about the folder structure can be found in the “Structure” section of the README file.

Conclusion

- Prompt Engineering Alone Is Not Enough
 - Natural language has ambiguity.
 - TaskLang provided necessary structure and standardization.
- TaskLang Emerged as the Most Powerful Tool
 - Enabled consistent graph generation.
 - Worked across multiple LLMs.
 - Made it possible to scale knowledge extraction.
- LLMs + Graphs = Symbiotic Strength
 - LLMs are creative and flexible but imprecise.
 - Graphs are structured and precise but hard to build manually.
 - Together, they allow for scalable, explainable knowledge systems.

Conclusion

- Major Insights
 - Using TaskLang and LLMs together makes knowledge graph generation scalable.
 - RAG and Neo4j's embedded LLM enable real-time, intelligent querying.
 - Even with the natural language noisy input, the system can retrieve relevant structured data.
- Project Goals Achieved
 - Built an end-to-end pipeline from unstructured recipe text to structured querying.
 - Demonstrated how structured and unstructured AI methods can coexist.
 - Open-sourced the work for transparency and future research.