Filtering Techniques

Collectively, these papers develop methods to tailor data and retrieval for edge-constrained scenarios across four domains—Roleplay, Function Calling, Robotics, and Retrieval-Augmented Generation (RAG). They span techniques from selective indexing and adaptive caching (EdgeRAG), synthetic data synthesis and filtering (TinyAgent, Unlocking Data Synthesis, RouteNator, ToolFlow), graph-guided retrieval for persona consistency (RoleRAG), hard-negative mining for embedding quality (NV-Retriever), progressive and iterative RAG adaptations for embodied tasks (P-RAG, RAG-Modulo), to reasoning-augmented multi-modal tuning for precise robotic outputs (RT-Grasp).

Article Summaries

EdgeRAG: Online-Indexed RAG for Edge Devices

Proposes **EdgeRAG**, which prunes second-level embeddings in an IVF index and generates them on-demand, while pre-computing and adaptively caching embeddings for large “tail” clusters to fit RAG pipelines within limited memory and meet latency targets on devices like NVIDIA Jetson Orin Nano.
https://arxiv.org/html/2412.21023v1

1. Indexing Phase

- **Data Chunking:** Incoming data is split into smaller, overlapping chunks.
- **Embedding Generation:** Each chunk is converted into a high-dimensional vector (embedding) using an embedding model.
- **Clustering:** Embeddings are grouped into clusters using a clustering algorithm (e.g., k-means).
- **Centroid Storage:** Only the centroids (representative vectors) of each cluster are kept in memory.

2. Selective Embedding Storage

- **Profiling Clusters:** During indexing, EdgeRAG measures how long it takes to generate embeddings for each cluster.
- **Tail Cluster Identification:** Clusters that take too long to generate embeddings (“tail clusters”) are identified.
- **Pre-computation:** Embeddings for these slow clusters are precomputed and stored on disk to avoid delays during retrieval.

3. Retrieval Phase (When a Query Arrives)

- **Query Embedding:** The user’s query is embedded into a vector.
- **Centroid Search:** The system finds the closest centroids to the query embedding.
- **Second-Level Search:**
  - For clusters with precomputed embeddings (tail clusters), EdgeRAG loads these from storage.
  - For other clusters, it generates the necessary embeddings on-the-fly.
- **Similarity Search:** The system compares the query embedding to the relevant data embeddings to find the best matches.

4. Adaptive Caching

- **Caching Frequently Used Embeddings:** EdgeRAG keeps recently or frequently used embeddings in memory, reducing redundant computation and speeding up repeated queries.
- **Cache Management:** The cache size is adjusted based on available memory and service requirements.

5. Response Generation

- **Data Retrieval:** The most relevant data chunks are retrieved based on similarity.
- **LLM Generation:** These chunks are fed into a local Large Language Model (LLM) to generate a response for the user.

TinyAgent: Function Calling at the Edge

Presents **TinyAgent**, an end-to-end framework for fine-tuning small LMs on a systematically curated function-calling dataset. It introduces tool retrieval to shorten prompts and applies quantization for real-time, on-device assistant deployment that rivals GPT-4-Turbo's function-calling accuracy. https://arxiv.org/html/2409.00608v1

1. Sanity Checks and Validation

- **Ensures Data Quality:** By verifying that each function-calling plan forms a feasible graph and uses correct function names and argument types, only valid and executable plans are included in the training data.
- **Prevents Model Confusion:** Filtering out plans with errors or inconsistencies helps the model learn correct patterns, reducing the risk of hallucinations or invalid outputs during inference.

2. Graph Isomorphism Check

- **Focuses on Structure, Not Just Content:** By requiring the generated plan's Directed Acyclic Graph (DAG) to be structurally identical to the ground truth, the model is trained to understand not just which functions to call, but also the correct order and dependencies.
- **Encourages Robust Reasoning:** This method ensures the model learns to orchestrate function calls in a way that matches real-world requirements, which is crucial for reliable automation on edge devices.

Unlocking the Power of Data Synthesis for Function Calling with Fine-Tuned SLMs

Describes a **multi-stage synthetic data pipeline** using a mid-tier LLM (Phi3.5 MoE) for edge use. It emphasizes rigorous format and semantic verification to filter out errors, yielding high-quality function-calling examples for fine-tuning SLMs under license constraints . https://medium.com/data-science-at-microsoft/unlocking-the-power-of-data-synthesis-for-function-calling-with-fine-tuned-slms-a733df7a1d07?utm_source=chatgpt.com

1. Format Checking

Purpose:

Ensures that each data entry strictly follows the required structure and parameter constraints.

How it works:

- Checks for required arguments.
- Validates argument types (e.g., string, number).
- Confirms enum values are valid.
- Ensures numeric values are within defined ranges.
- Verifies conditional requirements.

Application to ClimbLab:

Use a format checker to automatically reject entries with missing fields, wrong types, or out-of-range values.

2. Semantic Checking

Purpose:

Verifies that the data semantically aligns with the intended meaning or task.

How it works:

- Sends both the user query and the associated data (e.g., function call or label) to a language model.
- Asks the model to judge if the data correctly addresses the query's intent.
- Uses a deterministic approach (temperature = 0) for consistent results.

Application to ClimbLab:

Use a semantic checker to filter out entries where the label or annotation does not accurately reflect the user's query or the data's context.

3. Deduplication

Purpose:

Removes duplicate entries to ensure dataset diversity and quality.

How it works:

- Tracks queries or data points.

- Keeps only the first occurrence of each unique entry.

Application to ClimbLab:

Run a deduplication step to eliminate repeated or near-identical data points.

4. Stratified Data Splitting (for Evaluation)

Purpose:

Ensures balanced representation of different classes or parameter combinations in training and validation sets.

How it works:

- Splits data based on function names and argument combinations.
- Maintains diversity and prevents overfitting to certain types.

RouteNator: Router-Based Multi-Modal Synthetic Data Generation

Introduces **RouteNator**, a router-driven architecture combining content metadata, knowledge graphs, and text/vision models to generate synthetic, distribution-matched function-calling datasets—improving classification accuracy and parameter selection in API orchestration tasks . https://arxiv.org/abs/2505.10495?utm_source=chatgpt.com

1. Metadata-Driven Filtering

- **Extract and analyze metadata** (e.g., content type, function type, query length, modality) from each ClimbLab data point.
- **Select samples** that match the real-world distributions needed for your target edge LM tasks (e.g., function calling, roleplay, robotics, RAG).
- **Balance** the dataset across content types and query types to avoid overfitting to dominant categories.

2. Distributional Matching

- **Analyze the distribution** of key features in real-world user queries (length, content type, keyword position, etc.).
- **Filter or reweight** ClimbLab samples so that your training subset closely matches these distributions, as RouteNator does with its weighted router.

- This helps your model generalize better and perform well on the challenge's evaluation benchmarks.

3. Semantic and Format Validation

- **Run format checks** to ensure all samples are structurally correct (e.g., valid function call formats, no missing fields).
- **Apply semantic validation** (possibly using a smaller LLM or rule-based system) to confirm that each function call or label accurately matches the user query's intent.
- **Remove or downweight** samples with ambiguous, incorrect, or low-quality mappings.

4. Deduplication and Diversity Enhancement

- **Identify and remove duplicates** or near-duplicates to maximize the diversity of your filtered dataset.
- **Prioritize samples** that introduce unique query structures, rare content types, or edge cases, as these improve model robustness.

5. Adaptive Filtering and Iterative Refinement

- **Continuously monitor** the statistics of your filtered dataset (e.g., after each filtering step).
- **Iteratively adjust** your filtering criteria to maintain the desired balance and diversity, using feedback from validation performance or pilot fine-tuning runs.

6. Documentation and Reproducibility

- **Document your filtering pipeline** clearly, including all rules, thresholds, and validation steps.
- This is required for the challenge and ensures your results can be reproduced and validated by the organizers.

1. Router-Based Multi-Modal Data Generation

RouteNator's core innovation is a **router-based architecture** that dynamically selects among multiple data generation strategies (heuristic, text-to-text LLM,

vision-to-text LLM) based on population-level statistics and content type requirements. This ensures:

- **Diversity**: Synthetic data covers a wide range of query types, content types, and modalities (text and images).
- **Realism**: The distribution of generated data closely matches real-world user queries, which is crucial for edge LMs to generalize well.

How to apply for the challenge:

- Implement a router that routes data generation requests to different prompt templates or models based on the target distribution of your downstream tasks (e.g., function calling, roleplay, RAG).
- Use both text and vision models if your edge LM tasks involve multi-modal data, as this increases the diversity and representativeness of your dataset.

2. Integration of Domain Knowledge and Metadata

RouteNator leverages **structured domain knowledge** (knowledge graphs, content metadata) to generate contextually relevant and semantically rich queries. This approach:

- Produces data that reflects real-world task complexity and intent.
- Ensures coverage of edge cases and rare content types, which is important for robust edge LM performance.

How to apply for the challenge:

- Extract and use metadata (titles, keywords, tags) and knowledge graphs relevant to your edge LM's target domains (e.g., mobile function calling, robotics).
- Generate synthetic queries by combining these elements, ensuring your dataset is both broad and deep in coverage.

3. Rigorous Validation and Filtering

RouteNator's pipeline includes **validation checks** for:

- Query realism (natural language quality)
- Label accuracy (correct function/API mapping)

- Distribution alignment (matching real-world query length, content type, and keyword position distributions)
- Deduplication and removal of unnatural or outlier queries

How to apply for the challenge:

- After generating synthetic data, apply multi-stage filtering:
  - **Format checking**: Ensure all data points conform to the required structure (e.g., valid JSON, correct parameter types).
  - **Semantic checking**: Use a model or rule-based system to verify that the generated function/API call matches the user query's intent.
  - **Distributional filtering**: Adjust your dataset to match the real-world statistics of your target use case (e.g., query length, content type frequency).
- This step is critical for maximizing Simprove - Sbase in the challenge, as it directly impacts the quality and utility of your filtered dataset.

4. Continuous Monitoring and Adaptive Filtering

RouteNator continuously monitors the data generation process, adjusting sampling and filtering to maintain the desired balance and diversity.

How to apply for the challenge:

- Implement feedback loops that analyze your filtered dataset's statistics and iteratively refine your filtering/generation parameters to optimize for the challenge's evaluation metrics (accuracy, efficiency, and coverage across all ELMB tasks).

5. Alignment with Challenge Requirements

The Data Filtering Challenge emphasizes:

- **Task-specific performance** (especially function calling, roleplay, robotics, RAG)
- **Efficiency** (datasets must be high-quality and compact, suitable for edge LMs)
- **Reproducibility** (submit code, filtered data, and checkpoints)

RouteNator's systematic, multi-stage filtering and data generation pipeline is well-suited for these requirements, as it produces datasets that are both diverse and precisely tailored to downstream tasks.

ToolFlow: Natural and Coherent Dialogue Synthesis for Tool-Calling

Presents **ToolFlow**, which uses graph-based sampling to select relevant tool combinations and planned generation for coherence. Multiple agents interactively synthesize dialogues, yielding diverse, context-rich tool-calling examples that enhance SFT on models like LLaMA-3.1-8B .https://arxiv.org/abs/2410.18447?utm_source=chatgpt.com

1. Graph-based Sampling

This technique constructs a graph where each node represents a tool (or data feature), and edges represent strong semantic or functional relationships (e.g., shared parameters or compatible outputs/inputs). Instead of random sampling, which can lead to irrelevant or incoherent data, Graph-based Sampling ensures that only highly relevant and interrelated data points (or tool combinations) are selected.

How it works:

- Build a graph using feature similarity (e.g., using Sentence-BERT embeddings for parameter/return value descriptions).
- Define edges based on a similarity threshold (e.g., cosine similarity > 0.82).
- Sample connected subgraphs (random walks) to select data/tool combinations that are more likely to be useful together.

Why it's effective for the Challenge:

- Increases the relevance and diversity of your dataset.
- Reduces noise and redundancy, which is crucial for edge LMs with limited resources.
- Ensures that filtered data is more likely to reflect real-world, multi-step, or multi-tool scenarios—key for tasks like function calling and robotics.

2. Planned-Generation (Dialogue/Data Planning)

Rather than generating data or dialogues in a single pass, this method first creates a high-level plan outlining the logical flow or sequence of actions/requests. The actual data is then synthesized to follow this plan, ensuring coherence and natural progression.

How it works:

- For each sampled data subset, generate a plan (e.g., a sequence of user requests, tool calls, and chitchat).
- Use this plan to guide the generation of the final data/dialogue, ensuring each step logically follows from the previous.
- Incorporate both task-relevant and “off-task” (e.g., chitchat, transitions) elements to mimic real-world usage.

Why it’s effective for the Challenge:

- Greatly improves the coherence and naturalness of the filtered dataset.
- Produces data that better matches real-world, multi-turn interactions—important for edge LMs in interactive or roleplay scenarios.
- Helps avoid overfitting to simplistic or single-turn data, boosting downstream task performance.

Systematic Process for the Challenge

1. **Construct a Feature/Tool Graph:** Encode your dataset’s features, tools, or actions using semantic embeddings. Build a graph based on similarity or functional relationships.
2. **Apply Graph-based Sampling:** Use random walks or subgraph sampling to select highly relevant, interconnected data points.
3. **Generate High-Level Plans:** For each sampled subset, create a logical plan outlining the sequence of actions, requests, or interactions.
4. **Synthesize Data According to Plans:** Generate the final dataset (dialogues, function calls, etc.) by following the plan, ensuring coherence and diversity.
5. **Quality Filtering:** Optionally, apply rule-based or model-based filters to remove low-quality or incoherent samples.

6. **Evaluate and Iterate:** Assess the filtered dataset's diversity, coherence, and downstream task performance; iterate to refine your filtering pipeline.

RoleRAG: Graph-Guided Retrieval for Role-Playing Agents

Develops **RoleRAG**, a retrieval framework that normalizes entity variants via semantic clustering and builds a character knowledge graph. Its boundary-aware retriever extracts role-consistent content and rejects out-of-scope queries, reducing hallucinations in multi-turn persona simulations .https://arxiv.org/html/2505.18541v1?utm_source=chatgpt.com

1. Semantic Entity Normalization

What it is:

RoleRAG introduces an entity normalization algorithm that merges duplicated or ambiguous names (e.g., "Anakin Skywalker" and "Darth Vader") into unified canonical entities. This is done by:

- Extracting entities and their descriptions from your dataset.
- Using vector embeddings to find semantically similar entities.
- Prompting an LLM to confirm if two entities are the same and then clustering them.
- Assigning a single canonical name to each cluster.

Why it matters for the challenge:

- Removes redundancy and ambiguity, ensuring your filtered dataset is clean and consistent.
- Reduces noise, which is crucial for edge LMs with limited resources.
- Improves retrieval and downstream model accuracy by making entity references unambiguous.

How to apply:

- Run entity extraction and normalization as a preprocessing step on your dataset before fine-tuning or training edge LMs.

- Use vector-based similarity search and LLM-based confirmation to cluster and unify entities.
- Normalize all references in your dataset to these canonical forms.

2. Boundary-Aware Retrieval Filtering

What it is:

RoleRAG's retrieval module is designed to:

- Extract only the information relevant to the target role or task.
- Explicitly reject or filter out data that falls outside the intended knowledge boundary (e.g., questions about Apollo 11 to an ancient figure).
- Use a knowledge graph to structure and filter context, ensuring only in-scope, high-quality data is used.

Why it matters for the challenge:

- Prevents hallucinations and out-of-scope responses, which is critical for edge LMs in real-world applications.
- Ensures that the filtered dataset is highly relevant to the downstream tasks (roleplay, function calling, robotics, RAG) specified in the challenge.
- Reduces dataset size by excluding irrelevant or misleading data, optimizing for the 10B token upper bound.

How to apply:

- Build or use a knowledge graph to represent your dataset's entities and their relationships.
- For each data point, check if it is within the scope of the target task or role; filter out anything that is not.
- Use LLMs to help assess the relevance and boundary of each data point, as in RoleRAG's retrieval module.

3. Contextual Chunking and Relation Extraction

What it is:

- Split large documents into manageable chunks.

- Extract not just entities, but also their relationships and context.
- Store these in a structured format (e.g., a knowledge graph) for efficient filtering and retrieval.

Why it matters for the challenge:

- Ensures that only the most relevant, context-rich data is retained.
- Supports efficient downstream retrieval and model training, which is essential for edge devices.

How to apply:

- Preprocess your dataset by chunking and extracting entities/relations.
- Use these structures to filter and organize your data for training.

4. Rejection of Out-of-Scope Data

What it is:

- Explicitly identify and remove data that is not relevant to the target use case or exceeds the knowledge boundary of the intended model/task.

Why it matters for the challenge:

- Directly aligns with the challenge's focus on improving model accuracy and applicability for edge LMs.
- Reduces the risk of model hallucination and irrelevant outputs.

NV-Retriever: Hard-Negative Mining for Text Embeddings

Proposes a family of **positive-aware hard-negative mining** methods to remove false negatives in contrastive fine-tuning of embedding models. The resulting NV-Retriever-v1 achieves state-of-the-art retrieval scores (60.9 on MTEB BEIR) by balancing training speed and accuracy . https://arxiv.org/abs/2407.15831?utm_source=chatgpt.com

What Is Positive-Aware Hard Negative Mining?

Positive-aware hard negative mining is a data filtering strategy used in training embedding models (like those for text or images) with contrastive learning. Its goal is to select challenging negative examples (hard negatives) for each query,

but to avoid including negatives that are actually very similar—or even identical—to the positive example (these are called false negatives).

Why Is This Needed?

- **Contrastive learning** works by pulling positive pairs (e.g., a query and its correct answer) closer in the embedding space, and pushing negative pairs (e.g., a query and an unrelated passage) further apart.
- If a negative is too similar to the positive (a false negative), the model gets a confusing signal and may learn the wrong relationships.
- Random negatives are often too easy, so hard negatives (negatives that are similar but not correct) are more useful—but only if they're not actually positives in disguise.

How Does Positive-Aware Hard Negative Mining Work?

Here's a systematic process:

1. **Compute Similarity Scores**
    - For each query, use a strong teacher model to compute similarity scores between the query and all candidate passages.
2. **Identify the Positive**
    - Find the passage that is the true positive (the correct answer or match for the query) and note its similarity score.
3. **Set a Threshold Based on the Positive**
    - Define a threshold for negatives, based on the positive's similarity score. There are two main ways:
        - **Absolute Margin:** Only keep negatives where
        - **Relative Margin (TopK-PercPos):** Only keep negatives where (e.g., 95% of the positive's score)
4. **Filter Negatives**
    - Discard any negative whose similarity score is too close to the positive's score (i.e., above the threshold). This helps remove false negatives.
5. **Select Hard Negatives**

- From the remaining candidates, select the top-k with the highest similarity scores (but still below the threshold). These are your hard negatives.

NV-Retriever-v1 Model Card (Hugging Face)

Documents **NV-Retriever-v1**, an embedding model based on Mistral-7B with bidirectional attention masking, featuring average pooling and tuned via hard-negative mining. It provides usage guidelines and evaluation results on MTEB benchmarksǴ .https://arxiv.org/abs/2407.15831?utm_source=chatgpt.com

P-RAG: Progressive RAG for Embodied Everyday Tasks

Introduces **Progressive Retrieval Augmented Generation (P-RAG)**, which iteratively updates a database of past task interactions (goal, scene graph, trajectory) and retrieves both similar tasks and situations. This progressive scheme improves planning in simulated embodied environments without ground-truth few-shots . https://arxiv.org/pdf/2409.11279?utm_source=chatgpt.com

1. Progressive, Iterative Data Filtering

P-RAG's core innovation is its progressive, iterative approach to data curation. Instead of filtering your dataset in a single pass, you should:

- **Iteratively update your filtered dataset** after each round of model interaction or evaluation.
- Use feedback from model performance (e.g., which data led to successful task completions) to refine your filtering criteria in subsequent rounds.
- This mirrors how P-RAG accumulates high-quality, task-relevant experiences over time, leading to continual improvement without relying on ground-truth labels.

2. Context-Aware and Granular Filtering

P-RAG doesn't just filter for task similarity; it also retrieves and prioritizes data based on **situation similarity** (e.g., scene graphs, environmental context):

- **Embed both the instruction and the context** (such as scene graphs or metadata) for each data point.

- When filtering, select data that is not only topically relevant but also contextually similar to the target use case (e.g., similar environment, object relationships, or user intent).
- This dual filtering (task + situation) ensures that your dataset is highly relevant for downstream edge LM tasks, especially in robotics, RAG, and interactive environments.

3. Use of Embedding-Based Similarity

P-RAG leverages embedding models (like MiniLM) to encode both instructions and context, then uses **cosine similarity** to retrieve the most relevant historical data:

- **Convert all candidate data and queries into embeddings** using a strong, efficient model.
- Use similarity search (e.g., in a vector database) to filter out data that is semantically or contextually distant from your target tasks.
- This approach is scalable and aligns with best practices in modern RAG and edge LM pipelines.

4. Quality and Error Filtering

P-RAG includes mechanisms to check for action validity and filter out low-quality or invalid outputs:

- **Implement automated quality checks** (e.g., regex validation, action space matching) to remove noisy, irrelevant, or malformed data.
- This step is crucial for edge LMs, where efficiency and accuracy are paramount.

5. Self-Iteration and Bootstrapping

P-RAG demonstrates that **self-iteration**—using the model's own outputs to further refine the dataset—can yield significant performance gains:

- After an initial round of filtering and model training, use the model's improved outputs to further filter and enhance your dataset.
- This bootstrapping process can be repeated until performance saturates, maximizing the value of your filtered data.

RAG-Modulo: Memory-Augmented Agents with Critics

Presents **RAG-Modulo**, enhancing LLM-based agents with an interaction memory and a bank of syntax/semantics/execution critics. It retrieves relevant past experiences as in-context examples, enabling continual learning from successes and failures in long-horizon robotic tasks .https://arxiv.org/html/2409.12294v1?utm_source=chatgpt.com

1. Memory-Based Filtering and Experience Storage

What RAG-Modulo Does:

RAG-Modulo stores and retrieves past interactions, including both successes and failures, to inform future decision-making. This memory is continually updated as the agent solves new tasks, allowing it to learn from experience and avoid repeating mistakes.

How to Apply for the Challenge:

- **Curate datasets that include annotated examples of both successful and failed interactions.**
- **Filter out redundant, irrelevant, or low-quality data by leveraging memory of past outcomes.**
- **Prioritize data that has been shown to improve model performance on edge tasks (e.g., robotics, function calling, RAG).**

This approach ensures your dataset is not just large, but also rich in actionable, diverse, and high-quality examples—key for edge LMs with limited resources.

2. Critic Feedback Integration

What RAG-Modulo Does:

It uses a set of critics to evaluate candidate actions generated by the language model, filtering out infeasible or suboptimal actions before they are executed.

How to Apply for the Challenge:

- **Incorporate critic-like filters in your data pipeline to automatically flag and remove low-quality, infeasible, or irrelevant samples.**

- **Use automated or human-in-the-loop critics to assess the feasibility and utility of data points, especially for robotics and function-calling tasks.**

This step helps ensure that only data likely to improve downstream task performance is retained, directly aligning with the challenge's goal of maximizing edge LM accuracy and efficiency.

3. Interaction-Level Retrieval and Filtering

What RAG-Modulo Does:

Instead of retrieving entire past trajectories, RAG-Modulo retrieves the most relevant individual interactions from a diverse set of tasks, providing richer and more targeted context for decision-making.

How to Apply for the Challenge:

- **Design your filtering pipeline to select the most relevant data at the interaction (fine-grained) level, not just at the document or trajectory level.**
- **Use semantic similarity (embedding-based retrieval) to identify and keep only those interactions most relevant to the target edge LM tasks.**

This technique increases the diversity and relevance of your dataset, which is especially important for generalization to unseen tasks—a key evaluation metric in the challenge.

4. Optimal Context Size and Noise Control

What RAG-Modulo Finds:

Performance improves as you increase the number of relevant interactions (K) retrieved, but too many can introduce noise and degrade results.

How to Apply for the Challenge:

- **Empirically determine the optimal number of examples to include for each task (typically 5–10), balancing informativeness and noise.**
- **Filter out less relevant or low-quality interactions to avoid overwhelming the model with distracting context.**

This ensures your filtered dataset is both concise and effective, maximizing the improvement metric used in the challenge.

5. Continuous Dataset Improvement

What RAG-Modulo Shows:

Even starting with no prior experience, the system improves as it accumulates new, high-quality interactions.

How to Apply for the Challenge:

- **Iteratively refine your dataset by incorporating feedback from model performance on validation tasks.**
- **Continuously update your filters and memory with new, high-impact examples as you test and retrain.**

RT-Grasp: Reasoning Tuning for Robotic Grasping

Proposes **Reasoning Tuning**, a two-phase output structure for multi-modal LLMs: a reasoning step followed by numerical grasp-pose prediction. Paired with the **Reasoning Tuning VLM Grasp dataset**, it adapts LLMs for precise, context-aware grasp predictions, validated on real hardware. https://arxiv.org/html/2411.05212v1?utm_source=chatgpt.com

1. Reasoning Tuning: Structured Data with Reasoning Phase

Core Idea:

RT-Grasp introduces a “reasoning phase” in its dataset, where each data sample includes not just the input (e.g., an image and instruction) and the target output (e.g., a grasp pose), but also an explicit, structured reasoning step. This phase describes the object’s attributes (type, shape, position) and the logic behind the grasping strategy before providing the numerical prediction.

How to Apply for the Challenge:

- When filtering or generating data for edge LMs, ensure each sample includes a reasoning step that explains the logic or context behind the answer, not just the answer itself.
- For robotics or function-calling tasks, this means including a rationale for each action or prediction, which helps the model generalize and reason better on unseen data.