ChatGPT

Updating Axolotl Config: Using `chat_template` for ShareGPT-Format Dataset

Background: Recent versions of Axolotl have deprecated the `type: sharegpt` dataset format in favor of a more flexible `type: chat_template` approach [1]. In practice, this means you should replace the old `type: sharegpt` stanza (with its `conversation: chatml` setting) with a `chat_template` configuration. This ensures your ShareGPT-style data (the `SE6446/MAGllama_Sharegpt` dataset) is parsed and formatted correctly, preserving the conversation structure and special tokens without degrading model performance.

YAML Dataset Stanza Example (ChatML Template)

Below is an example **YAML** configuration snippet for the dataset, updated to use `chat_template`. This assumes the dataset entries have a top-level key `"conversations"` containing a list of messages, each with `"from"` and `"value"` fields (as is the case for ShareGPT-format data). We include all relevant options to maintain formatting consistency and training behavior:

```
chat_template: chatml   # Use ChatML template for formatting multi-turn
conversations (OpenAI Chat format)

datasets:
  - path: SE6446/MAGllama_Sharegpt
    type: chat_template        # New dataset type (replaces deprecated
'sharegpt')
    field_messages: conversations
# Key in each data sample that holds the list of message dicts [2]
    message_property_mappings:
# Map the dataset's message fields to the template's expected fields [2]
      role: from           # The dataset uses "from" (e.g. 'human', 'gpt',
'system') to indicate speaker role
      content: value       # The dataset uses "value" to hold the message
content text
    roles:                              # Map dataset-specific role labels to
the standard roles used in the template [3]
      assistant:
        - gpt              # Treat any "from: gpt" as an assistant role (the
AI's responses) [3]
      user:
        - human            # Treat any "from: human" as a user role (the human
prompts) [3]
```

```
# (No mapping needed for "system" – it's already named "system", which the
ChatML template will recognize as a system role)
    roles_to_train: ["assistant"]       # Only train on the assistant's
responses; user/system prompts are masked out [4] [5]
    train_on_eos: "turn"                # Only train on end-of-turn tokens for
assistant messages (skip EOS tokens after user/system turns) [5]

special_tokens:
  eos_token: "<|im_end|>"   # Define the ChatML end-of-message token as the EOS
token (ChatML uses <|im_end|> to terminate each turn) [6]
  additional_special_tokens: ["<thinking>", "</thinking>", "<reflection>", "</
reflection>", "<output>", "</output>"]  # Ensure custom tags are recognized as
single tokens
pad_token: "<|end_of_text|>"  # Use the end-of-text token as padding (to align
with Llama tokenizer defaults) [7]
```

Notes on the example: In this snippet, we explicitly specify the chat template (`chatml`) and map the dataset's structure to it. The `path` should remain pointing to **SE6446/MAGllama_Sharegpt**, and we preserve special tokens and padding settings from the original config.

Explanation of Key Fields and Format Requirements

- `type: chat_template` – This new type activates Axolotl's Jinja2-based chat prompt templating. It replaces the legacy `type: sharegpt` (which is no longer supported) as a drop-in solution for conversation datasets [1]. By using `chat_template`, we ensure the model sees prompts in the same format it will use during inference (in this case, OpenAI's ChatML style [2]).

- `chat_template: chatml` – Specifies the *formatting template* to use. Here we choose the **ChatML** template, since the original config used `conversation: chatml`. ChatML is the chat format with `<|im_start|>` and `<|im_end|>` tokens similar to OpenAI's API. Setting `chat_template: chatml` ensures the system, user, and assistant messages are wrapped with these tokens exactly as expected [2]. (If the base model's tokenizer had a built-in chat template, one could use `chat_template: tokenizer_default` or a fallback, but explicitly setting `chatml` guarantees consistency with the ShareGPT data format.)

- **Mapping Dataset Fields** – The ShareGPT-style dataset uses non-standard keys (`"from"` for speaker role and `"value"` for content). We use `field_messages: conversations` to tell Axolotl which field contains the list of messages, and `message_property_mappings` to map the internal keys to the template's expected keys [2]. In other words:

- `"from"` → **role** (e.g. `"human"` or `"gpt"` will be interpreted as the message role)
- `"value"` → **content** (the actual text content of each message)

These mappings are crucial; if they are omitted or incorrect, Axolotl may throw a KeyError or fail to format the conversation properly [8] [9]. The official Axolotl docs confirm that older ShareGPT-format data (using

`"conversations": [..., {"from": ..., "value": ...}, ...]`) should be handled by specifying these mappings [9].

- `roles` **Mapping** – We provide a mapping for role names to standardize them. The dataset uses `"human"` and `"gpt"` to denote the user and assistant, respectively. Axolotl's chat templates expect **user** and **assistant** roles (and optionally **system**). We therefore map:
- **assistant**: `["gpt"]` (and any other synonyms, if present, like `"model"`) [3]

- **user**: `["human"]` [3]
  This ensures that any `"from": "gpt"` entry in the dataset is treated as an assistant message, and `"from": "human"` as a user message. (The `"system"` role in the data is already named *system*, which the ChatML template will handle as a system message by default, so we don't need to remap it.) If you forget to include role mappings for custom labels, the template might not recognize them – leading to missing prompts or an error – so it's good practice to add these for non-standard role names [8].

- `roles_to_train: ["assistant"]` – This setting tells Axolotl to *only* generate training labels for the assistant's responses [4]. All non-assistant messages (user and system prompts) will be masked out (labelled `-100`), meaning the model won't be trained to predict those parts of the conversation. This is important for performance: we want the model to learn to produce the assistant's replies, not to mimic user queries or system instructions. By default, Axolotl's chat template logic focuses on training the assistant role, but we explicitly specify it for clarity. (If you wanted to train on other roles as well, you'd list them here and adjust the `roles` mapping accordingly.)

- `train_on_eos: "turn"` – This option controls how end-of-sequence (EOS) tokens are handled in training. `"turn"` means the model will only be trained on EOS tokens that mark the *end of a turn*, and will ignore EOS tokens at the ends of user/system messages [5]. In the ChatML format, each message (including user and assistant) is terminated with a special `<|im_end|>` token. With `train_on_eos: "turn"`, Axolotl will mask out those end-of-turn tokens for non-assistant turns (so the model doesn't try to predict when the user's message should end, for example) [5]. This prevents the model from unnecessarily learning to output EOS tokens for user prompts during training, which could degrade its conversational quality. In short, it only learns to predict the end-of-turn for its own (assistant) replies. This setting is recommended in multi-turn chat training to avoid confusing the model [5].

- **Special Tokens Configuration** – We include `special_tokens` in the config to align the tokenizer with the ChatML format and our dataset's custom tokens:

- **End-of-Turn/EOS Token:** The ChatML template uses the token `<|im_end|>` to signify the end of each message turn. It's important that the tokenizer treats this as an end-of-sequence token. We explicitly set `eos_token: "<|im_end|>"` so that Axolotl knows to use `<|im_end|>` as the EOS (instead of, say, the default `</s>` or `<|end_of_text|>` from the base model) [6]. The Axolotl documentation **highly recommends** doing this when your chat template's end-of-turn token differs from the model's default EOS [6] [10]. This way, the model will see `<|im_end|>` as the proper sequence terminator in context, matching how the prompt is constructed.
- **Additional Special Tokens:** The dataset includes tokens like `<thinking>`, `</thinking>`, `<reflection>`, `</reflection>`, `<output>`, `</output>` embedded in the assistant

responses (these appear to be special markup for the assistant's thought process, reflection, and final answer). We add all of these to `additional_special_tokens` so that the tokenizer knows they are single, indivisible tokens. This prevents them from being broken into subwords and ensures the model can generate them exactly. Preserving these tokens is crucial because they likely have semantic meaning in the fine-tuning context (e.g. the model was trained to use `<thinking>` ... `</thinking>` for chain-of-thought reasoning). By listing them here, we maintain compatibility with the model's vocabulary and avoid any unintended formatting loss. (Make sure the base model's tokenizer has been extended with these tokens – it appears **SuperNova-Lite** was indeed configured with them, given that your current config already includes them.)

- **Padding Token:** We set `pad_token: "<|end_of_text|>"` (the end-of-text token) as the padding token. Many LLaMA-based models don't have a dedicated pad token, so using the EOS token or a similar special token for padding is common [7]. This matches the original config and ensures that padding does not introduce an out-of-vocab token. By keeping `<|end_of_text|>` as `pad_token`, we avoid any shape or ID mismatches during training (Axolotl often requires a pad token to be defined for certain training routines).

Validation against Official Axolotl Documentation

The configuration above is derived from the latest Axolotl guidelines for conversation datasets. Axolotl's **"Migrating from sharegpt"** section confirms that a ShareGPT-formatted dataset (with `"conversations": [{"from": ..., "value": ...}, ...]`) should be configured using `type: chat_template` with `field_messages` and property mappings for role/content [9]. The snippet we provided aligns with that guidance, simply substituting your dataset path and adding the role mappings for completeness. The Axolotl docs also emphasize training only on assistant outputs and setting the appropriate end-of-turn EOS token for ChatML [4] [6] – both of which are reflected in our example.

Finally, the deprecation of `type: sharegpt` is well-documented: developers have noted that you should "set it to `type: chat_template` instead" [1]. By updating your YAML as shown and keeping all format-related options in place, you ensure that the `SE6446/MAGllama_Sharegpt` data is ingested exactly as intended, without any inadvertent changes to tokenization or prompt structure that could impact model performance. Always double-check the Axolotl release notes or config reference when upgrading, but this format should be a drop-in replacement that preserves the behavior of the old configuration while using the current, supported syntax.

Sources:

- Axolotl Documentation – *Conversation Dataset Formats* (official guide on using `chat_template` and mapping ShareGPT-style data) [9] [3] [6]
- Axolotl Issue Tracker – Deprecation notice for `type: sharegpt` (instructs to use `chat_template` in newer versions) [1]
- Example Axolotl Configs – User-contributed configs showcasing the new `chat_template` syntax with ShareGPT datasets and special tokens [7].

---

1 7 huggingface.co
https://huggingface.co/shisa-ai/ablation-26-reasoning.inputs-shisa-v2-llama-3.1-8b-lr8e6/resolve/main/README.md?download=true

2 9 10 Conversation – Axolotl
https://axolotl-ai-cloud.github.io/axolotl/docs/dataset-formats/conversation.html

3 4 5 6 8 Dataset Formats – Axolotl
https://axolotl-ai-cloud.github.io/axolotl/docs/dataset-formats/