



Exceptional service in the national interest

HANDS-ON WITH ATLAS/SHIRTY AS A TRILINOS DEVELOPER

Coding with LLMs at Sandia

Chris Siefert, Christian Glusa & Jonathan Hu

Trilinos Users Group Meeting, October, 2024



Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND 2024-14336PE

PRE-TALK PREPARATION FOR ATLAS/SHIRTY HANDS-ON



- Sign up for the *wg-atlas-users* at metagroup.sandia.gov. This will take ~15 minutes to process.
- In the Linux/Mac environment you're using, make sure your proxies are setup correctly.
- Find Chris, Christian or Jonathan if you're having trouble.

WHAT ARE ATLAS AND SHIRTY?



- ATLAS is an SRN SNL-hosted web-based LLM service.
 - Current models as of 10/1/24: Llama 3.1 70B, Mistral 8x7b, Mistral-7b
 - ATLAS does *not* log conversations, but does log metadata (e.g. which models used).
 - Incidental personal use of ATLAS is allowed.
- Besides off-the-shelf chat models, ATLAS also supports Retrieval Augmented Generation (RAG) with select SNL datasets (e.g. LDRD datasheets)
 - RAG = Do a search; feed results to LLM; let LLM summarize
 - ATLAS RAG also returns links to the relevant documents.
 - Be aware: Some datasets are marked as containing CUI.
- If you're in the right group, atlas.sandia.gov should work for you.



ATLAS CHAT EXAMPLE



You 10:57 AM

What does Sandia National Laboratories do for the nation?



ATLAS 10:57 AM

Sandia National Laboratories conducts research and development in national security and nuclear science, primarily for the United States Department of Energy's National Nuclear Security Administration (NNSA).



ATLAS RAG EXAMPLE (FINDING MY OLD ASC PROJECT)



You 11:03 AM

Describe Sandia work on linear solvers.



ATLAS 11:03 AM

Sandia is working on a project, "Machine Learning for Linear Solvers," led by Chris Siefert. The project aims to use machine learning methods to build predictive models for identifying optimal non-physics parameters, specifically linear solver and preconditioner settings, as a function of the algebraic and statistical properties of the linear operators involved in simulations. The goal is to improve the robustness and scalability of linear solvers, particularly for applications such as ice-sheet modeling and harmonic balance frequency domain circuit simulation.



» **References** 11:03 AM

[Reference Text](#) | [2019.pdf](#) | [2017.pdf](#) | wiki.sandia.gov/display/ASCML/Machine+Learning+for+Linear+Solvers

Include reference markings ⓘ



CUI markings will be included by default if a CUI RAG database is used

WHAT IS SHIRTY?



- Shirty is the SRN SNL toolkit for AI microservices. As of 10/1/24, the supported models are:
 - Speech-to-text: Sstream/faster-whisper-tiny, Systran/faster-whisper-medium, Systran/faster-distil-whisper-medium.enm Systran/faster-whisper-large-v3, System/faster-distil-whisper-large-v3
 - Text-to-speech: tts-1
 - LLM: codeqwen, thesven/Mistral-7B-Instruct-v0.3-GPTQ, neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8, NousResearch/Meta-Llama-3-8B-Instruct, dophin-2.7-mixtral-8x7b-AWQ
 - Vision LLM: Phi-3-vision-128k-instruct
- All instructions we're using can also be found at:
<https://shirty.sandia.gov/index.html>



OHPC Shirty is coming, but is not here yet!

GET A SHIRTY API KEY

- Go here: <https://shirty.sandia.gov/start/>



API Key Request Form

PT (for tracking)

Submit

- You will need a P/T for tracking purposes only. It will not be charged.
- This will give you your API key. Save this!



VIM USERS: HOW TO SHIRTY



- Use up-to-date vim on CEE:

```
module use /projects/trilinos/modulefiles
module load vim91
```

- Get the AI plug-in

```
mkdir -p ~/.vim/pack/plugins/start
git clone https://github.com/madox2/vim-ai.git ~/.vim/pack/plugins/start/vim-ai
```

- Modify ~/.vimrc:

```
let chat_engine_config = {
\ "engine": "chat",
\ "options": {
\   "model": "neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8",
\   "endpoint_url": "http://shirty.sandia.gov/api/v1/chat/completions",
\   "max_tokens": 0,
\   "temperature": 0.1,
\   "request_timeout": 20,
\   "selection_boundary": "",
\   "key": "YOUR_KEY_HERE",
\   "initial_prompt": "",
\ },
\}
let g:vim_ai_complete = chat_engine_config
let g:vim_ai_edit = chat_engine_config
```

- Fun Vim Shirty commands

<ctrl-v> to highlight particular code

:AI document this code

:AI explain this code

:AI rewrite this code

:help vim-ai

For more information, see:

<https://github.com/madox2/vim-ai>

EMACS USERS: HOW TO SHIRTY



- If you're on CEE, get a new emacs:

```
module use /projects/emacs/modules
```

```
module load emacs
```

- Download the ELPA keyring update

```
wget https://elpa.gnu.org/packages/gnu-elpa-keyring-update-2022.12.1.tar
```

- Download the appropriate branch of GPTEL

```
git clone https://github.com/janEbert/gptel.git
```

```
(cd gptel; git fetch origin; git checkout --track  
remotes/origin/optional-system)
```

- Start emacs and install the keyring

```
M-x package-install-file (ENTER) gnu-elpa-  
keyring-update-2022.12.1.tar
```

- Install transient

```
M-x package-install (ENTER) transient
```

- Install GPTEL

```
M-x package-install-file (ENTER) gptel
```

- Add the following to your ~/.emacs

```
(setq
```

```
gptel-model "neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8"
```

```
gptel-backend (gptel-make-openai "shirty"
```

```
  :stream t
```

```
  :protocol "https"
```

```
  :host "shirty.sandia.gov"
```

```
  :endpoint "/api/v1/chat/completions"
```

```
  :key "YOUR_KEY_HERE"
```

```
  :models '("neuralmagic/Meta-Llama-3.1-70B-Instruct-  
FP8")))
```

```
(setq gptel-api-key "YOUR_KEY_HERE")
```

```
;;(setq gptel-log-level 'debug) ;; This is useful if things don't  
work
```

```
(setq gptel-system-message nil) ;; Mistral doesn't use the  
"system message"
```

```
````
```



## EMACS USERS: HOW TO SHIRTY

- You can directly execute the code by placing the cursor after the snippet and `C-x C-e`.
- Run to open a dedicated chat buffer `M-x gptel`
- Write your query after `###` and submit via `C-c RET`.
- `C-u C-c RET` instead opens a menu that allows to modify parameters for the query (model, context, system prompt, etc).
- Another way of interacting with an LLM is directly from another buffer. E.g. in a code buffer one can select a region that should be used for the query and do `M-x gptel-send`.

Finally, marking a region and calling `C-u M-x gptel-send` and the hitting `r` for "refactor" requests a refactor of the selected region. Once the query is processed the changes can be displayed as diff, accepted or rejected via the key combos displayed at the bottom of the screen.

- You can also add files or regions to the context for the LLM and ask questions about them. Use `M-x gptel-add` to add a region or `M-x gptel-add-file` to add a file. Then query using `C-u M-x gptel-send` and select the context by typing `C`.
- More information can be found here: <https://github.com/karthink/gptel>



# VSCODE USERS: HOW TO SHIRTY

- Install the continue.dev extension in vscode.
- Ensure you have the sandia ssl cert somewhere, modify the caBundlePath keys to point to it
- Edit the config by running Ctrl+Shift+P and then finding the Continue: Open config.json command and selecting it and copy in the following...

```
{
 "tabAutocompleteModel": {
 "title": "Shirty",
 "provider": "openai",
 "model": "neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8",
 "apiKey": "YOUR_KEY_HERE",
 "apiBase": "http://shirty.sandia.gov/api",
 "requestOptions": {
 // "caBundlePath": "path/to/your/cert/nix-new.crt",
 "verifySsl": true,
 "noProxy": [".sandia.gov", "sjirty.sandia.gov"]
 }
 },
 "models": [
 {
 "title": "Shirty",
 "provider": "openai",
 "model": "neuralmagic/Meta-Llama-3.1-70B-Instruct-FP8",
 "apiKey": "YOUR_KEY_HERE",
 "apiBase": "http://shirty.sandia.gov/api",
 "requestOptions": {
 // "caBundlePath": "path/to/your/cert/nix-new.crt",
 "verifySsl": true,
 "noProxy": [".sandia.gov", "shirty.sandia.gov"]
 }
 }
],
 "allowAnonymousTelemetry": false,
 "slashCommands": [
 {
 "name": "edit",
 "description": "Edit selected code"
 },
 {
 "name": "comment",
 "description": "Write comments for the selected code"
 },
 {
 "name": "share",
 "description": "Download and share this session"
 },
 {
 "name": "cmd",
 "description": "Generate a shell command"
 }
]
}
```

## VSCODE USERS: HOW TO SHIRTY



- Select the code you care about and `Ctrl+L` to add to chat.
  - "Explain this code in plain english." is something fun to try.
  - The `/comment` slash command is another option.
  
- Slash commands for `continue.dev` are documented here:  
<https://docs.continue.dev/customize/slash-commands>