

The background features a dynamic, abstract pattern of blue dots and lines that form a series of overlapping, wavy shapes, resembling a digital signal or a stylized wave. This pattern is set against a dark blue gradient background.

# HELMHOLTZ AI

## **CODING ASSISTANTS AND HOW TO SET THEM UP**

Workshop - Level Up Your Skills

Till Korten

Helmholtz-Zentrum Dresden-Rossendorf / September 9, 2025

# WHAT ARE CODING ASSISTANTS?

src > cards > unet.py > DoubleConv

```
143
144 Alindat0er0ek, 2 days ago | 3 authors (Alindat0er0ek and others)
145 class SegmentationModel(pl.LightningModule):
146
147     def __init__(
148         self, in_channels=2, out_channels=2, lr=2e-4, lr_scheduler_patience=4, lr_scheduler_factor=0.5, **kwargs
149     ):
150         super().__init__()
151         self.model = UNet(input_channels=in_channels, output_channels=out_channels, **kwargs)
152         self.lr = lr
153         self.lr_scheduler_patience = lr_scheduler_patience
154         self.lr_scheduler_factor = lr_scheduler_factor
155
156     def forward(self, x):
157         return self.model(x)
158
159     def dice_loss(self, pred, target, smooth=1.0):
160         intersection = (pred * target).sum(dim=2).sum(dim=2)
161         loss = 1 - (
162             (2.0 * intersection + smooth) / (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2) + smooth)
163         )
164         return loss
165
166     def mean_dice_loss(self, pred, target, smooth=1.0):
167         pred = nn.Sigmoid()(pred)
168         pred = pred.contiguous()
169         target = target.contiguous()
170         loss = self.dice_loss(pred, target, smooth)
171         return loss.mean()
172
173     def training_step(self, batch, batch_idx):
174         x, y = batch
175         logits = self(x)
176         loss = self.mean_dice_loss(logits, y)
177         self.log("train_loss", loss, on_step=True, on_epoch=True, prog_bar=True)
178         return loss
179
180     def validation_step(self, batch, batch_idx):
181         x, y = batch
182         logits = self(x)
```



- IDE-integrated tools that suggest code, document, refactor, explain, and generate tests.
- Powered by large language models (LLMs) and repository/IDE context.
- Accelerate routine tasks; you stay in control.  
Human-in-the-loop is essential.

# TYPICAL USE-CASES

**Inline code completion** Suggests code as you type, anticipating common patterns and completing statements.

**Docstrings and comments** Generates comprehensive documentation that follows project conventions.

**Test generation** Creates unit tests with appropriate assertions based on implementation code.

**API usage examples** Demonstrates how to use libraries and frameworks with contextually relevant examples.

# TYPICAL USE-CASES

**Boilerplate generation** Creates repetitive scaffolding code like class definitions, function templates, and data structures.

**Refactoring suggestions** Identifies code that could be improved and offers cleaner implementations, e.g. remove redundant code.

**Code explanation** Provides natural language descriptions of complex code for easier onboarding.

**Context querying** Answers questions about project structure, dependencies, and implementation details.



# INLINE CODE COMPLETION

src > cardo > unet.py > DoubleConv

```
143
144 Alinda@Geroek, 2 days ago | 3 authors (Alinda@Geroek and others)
145 class SegmentationModel(pl.LightningModule):
146
147     def __init__(
148         self, in_channels=2, out_channels=2, lr=2e-4, lr_scheduler_patience=4, lr_scheduler_factor=0.5, **kwargs
149     ):
150         super().__init__()
151         self.model = UNet(input_channels=in_channels, output_channels=out_channels, **kwargs)
152         self.lr = lr
153         self.lr_scheduler_patience = lr_scheduler_patience
154         self.lr_scheduler_factor = lr_scheduler_factor
155
156     def forward(self, x):
157         return self.model(x)
158
159     def dice_loss(self, pred, target, smooth=1.0):
160         intersection = (pred * target).sum(dim=2).sum(dim=2)
161         loss = 1 - (
162             (2.0 * intersection + smooth) / (pred.sum(dim=2).sum(dim=2) + target.sum(dim=2).sum(dim=2) + smooth)
163         )
164         return loss
165
166     def mean_dice_loss(self, pred, target, smooth=1.0):
167         pred = nn.Sigmoid()(pred)
168         pred = pred.contiguous()
169         target = target.contiguous()
170         loss = self.dice_loss(pred, target, smooth)
171         return loss.mean()
172
173     def training_step(self, batch, batch_idx):
174         x, y = batch
175         logits = self(x)
176         loss = self.mean_dice_loss(logits, y)
177         self.log("train_loss", loss, on_step=True, on_epoch=True, prog_bar=True)
178         return loss
179
180     def validation_step(self, batch, batch_idx):
181         x, y = batch
182         logits = self(x)
```

## Tipps:

- **Always review suggestions!** They may be incorrect or insecure.
- Use speaking function/variable names.
- Use comments to prompt for specific logic.
- Use Ctrl + → to partially accept suggestions.

The image shows a document page with extensive red and blue annotations. The text is in a serif font, and the annotations include underlines, highlights, and marginal notes. The page is numbered '1' in the top right corner. The annotations are dense and cover most of the page, indicating a thorough review or editing process. The red annotations are more prominent, often underlining words or phrases, while the blue annotations are used for additional notes or corrections. The overall appearance is that of a handwritten or typed document that has been heavily annotated for review or editing.

- Use existing docstrings as style examples.
- Ask for specific formats (e.g., RST, NumPy, Google).
- Edit and use inline completion if the first attempt is not satisfactory.
- Add documentation style to system prompt.

# TEST GENERATION

```
src > carde > unet.py > SegmentationModel > forward
9
You, 10 minutes ago | 3 authors (You and others)
10 class DoubleConv(nn.Module):
11     """
12     A module that implements a sequence of convolutional layers with batch normalization and ReLU activation.
13
14     This module creates a sequential model of convolutional layers where the number of channels
15     linearly interpolates from the input channels to the output channels across the specified number of layers.
16
17     Parameters:
18         input_channels : int
19             Number of input channels.
20         output_channels : int
21             Number of output channels.
22         num_layers : int
23             Number of convolutional layers in the sequence.
24
25     Returns:
26         torch.nn.Module
27             A sequential model containing the specified convolutional layers.
28
29     Example:
30         >>> double_conv = DoubleConv(64, 128, 2)
31         >>> x = torch.randn(1, 64, 32, 32)
32         >>> output = double_conv(x)
33         >>> output.shape
34         torch.Size([1, 128, 32, 32])
35
36 """
37
38 def __init__(self, input_channels, output_channels, num_layers):
39     super().__init__()
40     self.conv = nn.Sequential()
41     n_channels = np.linspace(input_channels, output_channels, num_layers + 1).astype(int)
42     for n in range(num_layers):
43         self.conv.add_module(f"conv{n}", nn.Conv2d(n_channels[n], n_channels[n + 1], 3, padding=1))
44         self.conv.add_module(f"batch_norm{n}", nn.BatchNorm2d(n_channels[n + 1]))
45         self.conv.add_module(f"relu{n}", nn.ReLU(inplace=True))
46
47 ...
```

## Tips:

- Add and refine docstrings to functions before generating tests.
- Review and adapt the first generated test to fit project style, then re-generate the rest.
- Use the re-generate function to get multiple variants.
- Sometimes the test file is in the wrong place. Just move it and it should work.

# API USAGE EXAMPLES

```
500  
501 # Get indices for the bin  
502 mask = (logits > lower) & (logits <= upper)  
503 bin_size = mask.sum()  
504  
505 if bin_size > 0:  
506     bin_confidence = logits[mask].mean()  
507     bin_accuracy = labels[mask].mean()  
508     ece += (bin_size.float() / logits.numel()) * torch.abs(bin_confidence - bin_accuracy)  
509  
510 return ece.item()  
511  
512  
513
```

```
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

prompt: write a function that loads a optuna study from a given path as pandas dataframe and plots the results using seaborn

- Often used libraries (Pandas, numpy, seaborn)
- Well documented open-source libraries.
- Test the generated code and refine as needed.

# BOILERPLATE GENERATION

```
76 train_loader, val_loader, test_loader = random_split(dataset, (train_size, val_size, test_size))
77
78 val_dataset.dataset.train = False
79 test_dataset.dataset.train = False
80
81 train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
82 (variable) test_loader = DataLoader (batch_size=batch_size)
83 test_loader = DataLoader(test_dataset, batch_size=batch_size)
84 return train_loader, val_loader, test_loader
85
```

## Tips:

- Be specific about requirements and constraints.
- Provide context: existing classes, interfaces, or data models.
- **Only use code that you understand.**

# REFACTORING SUGGESTIONS

```
99 \begin{frame}{What are coding assistants?}
100
101 \item Accelerate routine tasks; you stay in control. \emph{Human-in-the-loop is essential.}
102 \end{itemize}
103 \end{column}
104 \end{columns}
105 \end{frame}
106
107 \begin{frame}{Typical use-cases}
108 \begin{columns}[T,totalwidth=\textwidth]
109 \begin{column}[0.5\textwidth]
110 \begin{itemize}
111 \item Inline code completion
112 \item Boilerplate generation
113 \item Refactoring suggestions
114 \item API usage examples
115 \end{itemize}
116 \end{column}
117 \begin{column}[0.5\textwidth]
118 \begin{itemize}
119 \item Test generation
120 \item Docstrings and comments
121 \item Code explanation for onboarding
122 \item Querying project context
123 \end{itemize}
124 \end{column}
125 \end{columns}
126 \end{frame}
127
128 \begin{frame}{Coding assistant examples}
129 \begin{table}[width=\textwidth]
130 \tbl_struct{
131 \tbl_header{
132 \toprule
133 \textbf{Tool} & \textbf{Type} & \textbf{Notes}
134 \midrule
135 \href{https://github.com/features/copilot}{GitHub Copilot} & Commercial & Extension for VS Code/Spyder/JetBrains et
136 \href{https://continue.dev}{Continue.dev} & Open-source & Extension for VS Code or JetBrains; Works
137 \href{https://cursor.com}{Cursor AI} & Commercial & Stand alone IDE with deep code agent inte
138 \bottomrule
139 \end{table}
140 \end{frame}
```

## Tips:

- Use prompt to specify the type of refactor (e.g., reduce redundancy, split into atomic functions).
- Review changes carefully; automated refactors may introduce bugs.
- Create unit tests before refactoring to catch bugs and ensure behavior is preserved.

# CODING ASSISTANT EXAMPLES

| Tool           | Type        | Notes                                                                                                                                                                                                                                           |
|----------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GitHub Copilot | Commercial  | Extension for VS Code/Spyder/JetBrains etc.; Various commercial models available. <a href="https://github.com/features/copilot">https://github.com/features/copilot</a>                                                                         |
| continue.dev   | Open-source | Extension for VS Code or JetBrains; Works with local models (LM Studio) and on premise hosted models (e.g. <a href="https://chat.fz-rossendorf.de">https://chat.fz-rossendorf.de</a> ). <a href="https://continue.dev">https://continue.dev</a> |
| Cursor AI      | Commercial  | Stand alone IDE with deep code agent integration. (Not used in practical part) <a href="https://cursor.com">https://cursor.com</a>                                                                                                              |

# CODING ASSISTANTS & PRIVACY

- Coding assistants need to send your data (code/questions etc.) to the model
- Data **will** be sent to third-party servers for commercial assistants
- Sometimes you can opt out of data sharing or training (for a price)
- Only use open-source or self-hosted solutions for sensitive code

| Tool           | Privacy    | Use-cases                                                                                                                                                                           |
|----------------|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GitHub Copilot | Medium     | Open source/proprietary code; Copilot claims that user requests are not used for training by default.                                                                               |
| continue.dev   | High       | Open source/Proprietary/confidential code                                                                                                                                           |
| Cursor AI      | Low/medium | Open source code/proprietary code; Optional privacy mode claims to not share data with others, but Cursor will store your data (to provide "features") but not use it for training. |



# ALIGN CODING ASSISTANTS TO YOUR PROJECT

Whenever possible, provide examples of your preferred style, the assistant will learn from them.

**Project profile prompt (drop into the assistant configuration):**

```
You are my pair programmer for a research codebase.  
Stack: Python 3.11, pytest, numpy, pandas; style: black + ruff.  
Non-negotiables: type hints, docstrings, 95% test coverage target.  
When unsure, ask; propose small, diffable changes with rationale.
```

*Keep this short and concrete. Reuse it as a system message or workspace note.*

# MODEL SELECTION

- Match model to task: Performance trade-offs between latency and accuracy
  - Code completion** Low-latency models; smaller context windows: Qwen3-coder, Claude Sonnet 4, GPT-4.1-copilot
  - Chat** Interactive models; medium context windows: GPT-5, Claude Sonnet 4, Qwen3-coder
  - Tool-use** Powerful tool models; large context windows: GPT-5, Claude Opus 4.1, Qwen3-coder
- Proprietary vs. open-source models:
  - Proprietary models (e.g., GPT, Claude) often lead in performance but may have privacy concerns
  - Open-source models (e.g., Qwen, Llama, Mistral) offer more control and can be self-hosted for sensitive data
  - Considering hardware and maintenance, self-hosting is the most expensive option

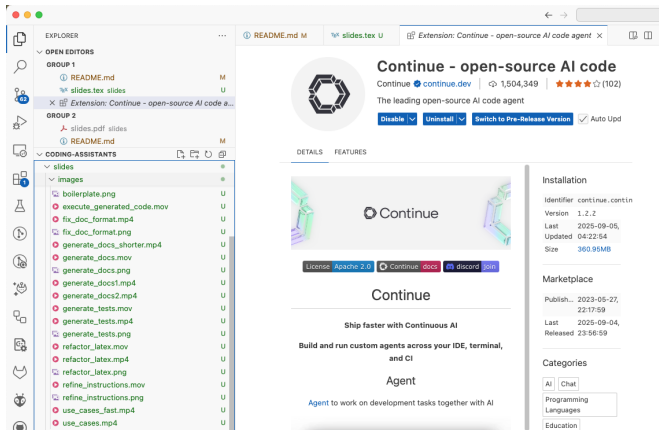
# LIVE CODING CHALLENGE: WHAT YOU WILL BUILD

- Small feature implementation with tests (language/IDE of your choice).
- Use assistant for: scaffolding, refactor steps, unit tests, docstrings.
- Measure: fewer keystrokes, faster iteration, equal or better quality.
- Project: your choice (e.g., personal, open-source, or provided sample).
- Example: write a function that loads a .csv file as pandas dataframe and plots the results using seaborn.

# SETUP - OPTIONS

- continue.dev with blablador (works everywhere)
- continue.dev with HZDR AI (works only on-campus (wired network or FSR wifi))
- continue.dev with LM Studio (on your machine, requires GPU with 20 GB RAM or Apple M CPU with  $\geq 24$ GB)
- GitHub Copilot (requires GitHub account)

## SETUP – ENVIRONMENT PREPARATION



- Install IDE extension (VS Code or JetBrains plugin).
- Create account/API Key.
- Configure the extension.
- Instructions on GitHub <https://github.com/thawn/coding-assistants/configurations/>

# EXAMPLE PROJECT

**Prompt:** write a python file that loads the diabetes dataset from scikit learn as pandas dataframe and plots the results using seaborn

Make sure the code:

- Loads the dataset using 'from sklearn.datasets import load\_diabetes'
- Converts it to a pandas dataframe
- Uses 'seaborn' to create a plot of the data

Next steps:

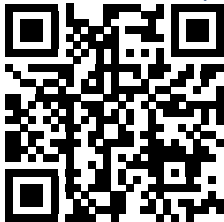
- Tell the assistant to convert the python file to a jupyter notebook
- Advanced: tell the assistant to convert the notebook to an installable package
- Advanced 2: Add docstrings and tests

# THANK YOU!

- The Helmholtz AI team
- Alex Strube and his team for Blablador
- Tobias Huste and his team for [chat.fz-rossendorf.de](https://chat.fz-rossendorf.de)
- Questions?

- Slides and example configurations:

DOI 10.5281/zenodo.17077277



- <https://github.com/thawn/coding-assistants/releases/download/v0.2.2/slides.pdf>