Documentation Complète

PatIA 120b

Assistant IA Expert C++11 pour Neovim

Version : 1.0
Date : 2025-11-01
Auteur : Claude Code
Modèle : OpenAI GPT-OSS 120B
Langue : Français

Configuration complète de Neovim avec OpenRouter

Table des matières

Chapitre 1

Présentation de PatIA 120b

1.1 Identité

PatIA 120b est un assistant IA expert intégré dans Neovim, spécialisé en développement C++11 moderne.

Nom	PatIA 120b
Modèle	OpenAI GPT-OSS 120B (120 milliards de paramètres)
Fournisseur	OpenRouter.ai
Langue	Français uniquement
Version Neovim	0.11 requise

1.2 Expertises

1.2.1 Programmation C++11

— Smart pointers (unique_ptr, shared_ptr, weak_ptr)
— RAII (Resource Acquisition Is Initialization)
— Move semantics et rvalue references
— Lambdas et fonctions anonymes
— Threading et concurrence (std : :thread, std : :mutex)
— Constexpr et compilation constante

1.2.2 Bibliothèques

— STL complète (conteneurs, algorithmes, itérateurs)
— libboost (SmartPtr, Thread, Filesystem, Asio, Regex)
— Gestion mémoire avancée

1.2.3 Système

— Administration Debian (toutes versions)
— Compilation et Make
— Debugging avec gdb
— Gestion de paquets (apt, dpkg)

1.3 Principes

1. **Smart pointers** plutôt que raw pointers
2. **RAII** pour toutes les ressources
3. **Move semantics** pour l'efficacité
4. **Const-correctness** systématique
5. **Code idiomatique** C++11
6. **Sécurité avant performance**

Chapitre 2

Installation

2.1 Installation Automatique

Installation Rapide

```
# Prerequis: Compte OpenRouter avec API Key
# https://openrouter.ai/

# Telecharger/copier les fichiers
cd /root/IA/NEOVIM

# Lancer l'installation
bash install.sh
```

Durée : 5-10 minutes
Le script installe :

- Neovim 0.11+ (via AppImage si nécessaire)
- lazy.nvim (gestionnaire de plugins)
- codecompanion.nvim
- Tous les plugins requis
- Configuration complète de PatIA 120b
- Script `ask-openrouter`

2.2 Vérification

Tests de Vérification

```
# Verifier l'installation
bash verify_installation.sh

# Test rapide
ask-openrouter "quel est ton nom ?"
# Doit repondre: "Je suis PatIA 120b..."

# Test dans Neovim
nvim test.cpp
:Ask bonjour
```

2.3 Correction Version Neovim

Si erreur `Neovim 0.11+ required` :

Correction Automatique

```
bash fix_neovim_version.sh
```

Choisir **option 1** (AppImage).

Chapitre 3

Commandes Neovim pour les LLM

Ce chapitre détaille **toutes** les commandes disponibles dans Neovim pour interagir avec PatIA 120b et les LLM.

3.1 Commandes Personnalisées

3.1.1 `:Ask <question>`

Description : Interroge PatIA en français (réponse garantie en français)
Syntaxe :

```
:Ask <votre question>
```

Exemples :

```
:Ask quel est ton nom ?
:Ask ecris une classe Vector3D en C++11
:Ask explique std::unique_ptr
:Ask comment utiliser std::move
:Ask donne-moi un exemple de RAII
```

Fonctionnement :

— Appelle le script `ask-openrouter` directement
— Utilise curl pour contacter OpenRouter API
— System prompt PatIA 120b intégré
— Réponses **TOUJOURS** en français
— Pas de contexte du fichier
— Affichage dans le terminal Neovim

Avantages

Français garanti
Rapide
Pas besoin de sélection

Inconvénients

- × Pas d'insertion automatique
- × Pas de contexte du code
- × Copie manuelle nécessaire

3.1.2 :AskCC <question>

Description : Alternative via CodeCompanion (peut répondre en anglais)
Note : Moins fiable pour le français, préférer `:Ask`

3.1.3 :AskBuffer

Description : Envoie tout le contenu du buffer au chat CodeCompanion
Fonctionnement :

1. Sélectionne tout le fichier (ggVG)
2. Ajoute au chat CodeCompanion
3. Ouvre la fenêtre de chat
4. Vous pouvez alors poser votre question sur le code

3.1.4 :Generate <prompt>

Description : Ouvre le chat avec un prompt de génération
Exemples :

```
:Generate une classe Shape avec polymorphisme
:Generate un singleton thread-safe en C++11
:Generate une factory de connexions TCP
```

3.1.5 :Translate <langage>

Description : Traduit le code actuel dans un autre langage
Exemples :

```
:Translate Python
:Translate Rust
:Translate Java
:Translate        " Par defaut: C
```

3.1.6 :MakeRun

Description : Compile avec make puis exécute si pas d'erreur

3.2 Raccourcis Clavier - Chat

3.2.1 `Espace c c` - Toggle Chat

Mode : Normal ou Visual
Description : Ouvre/ferme la fenêtre de chat CodeCompanion

3.2.2 `Espace c a` - Actions sur Sélection

Mode : Normal ou Visual
Description : Ouvre le menu d'actions CodeCompanion
Utilisation :

1. Sélectionner du code (mode VISUAL : V ou Shift+v)
2. Appuyer sur `Espace c a`
3. Choisir une action dans le menu

Action	Description	Résultat
Custom prompt	Question personnalisée	Réponse dans le chat
Fix code	Corriger le code	Code remplacé automatiquement
Explain	Expliquer le code	Explication dans le chat
Unit test	Générer tests unitaires	Tests générés
Add comments	Ajouter commentaires	Code commenté
Refactor	Refactoriser	Code amélioré
Optimize	Optimiser	Code optimisé

TABLE 3.1 – Actions disponibles avec Espace c a

3.2.3 `Espace c s` - Envoyer Message (Normal)

Mode : Normal (dans le chat)
Description : Envoie le message dans le chat CodeCompanion

3.2.4 `Ctrl+s` - Envoyer Message (Insert)

Mode : Insert (dans le chat)
Description : Envoie le message en cours de frappe

3.2.5 `Espace c g` - Générer Code

Mode : Normal
Description : Raccourci pour `:Generate`

Raccourci	Action
`Espace m m`	Make (compiler)
`Espace m c`	Make clean
`Espace m t`	Make target
`Espace m e`	Ouvrir liste erreurs
`Espace m q`	Fermer liste erreurs
`Espace m n`	Erreur suivante
`Espace m p`	Erreur précédente
`Espace m r`	Make & Run
`F2`	Script compilation custom
`F3`	Script sauvegarde custom

TABLE 3.2 – Raccourcis de compilation

3.3 Raccourcis Clavier - Compilation

3.4 Modes d'Interaction

3.4.1 Mode 1 : Questions Directes (`:Ask`)

Quand utiliser :

— Questions rapides

— Pas besoin de contexte du fichier

— Réponse en français garantie

3.4.2 Mode 2 : Chat Interactif (`Espace c c`)

Quand utiliser :

— Conversation longue

— Questions de suivi

— Besoin de contexte conversationnel

3.4.3 Mode 3 : Actions sur Code (`Espace c a`)

Quand utiliser :

— Modifier du code existant

— Besoin de remplacement automatique

— Fix, refactor, optimize

3.4.4 Mode 4 : Génération de Code (`:Generate`)

Quand utiliser :

— Créer du nouveau code

— Prototypes de classe

— Fonctions complexes

3.5 Tableau Récapitulatif

Commande	Mode	Action	Insertion Auto
`:Ask`	Normal	Question directe	×
`:AskCC`	Normal	Question via CC	×
`:AskBuffer`	Normal	Envoie fichier	×
`:Generate`	Normal	Génération code	×
`:Translate`	Normal	Traduction code	×
`Espace c c`	Normal/Visual	Toggle chat	×
`Espace c a`	Normal/Visual	Actions sélection	
`Espace c s`	Normal	Envoyer message	×
`Ctrl+s`	Insert	Envoyer message	×
`Espace c g`	Normal	Générer code	×

Chapitre 4

Utilisation Complète

4.1 Scénario 1 : Créer une Classe C++11

Objectif : Créer une classe Point3D avec smart pointers
Méthode A - Commande directe :

```
:Ask ecris une classe Point3D avec std::unique_ptr
```

Résultat :

- Code complet avec `#include`
- Commentaires en français
- Explications RAII
- Commande de compilation

4.2 Scénario 2 : Corriger du Code Bugué

Code avec bug :

```
void process(int* ptr) {
    *ptr = 42;  // Pas de verification nullptr
}
```

Correction :

1. Sélectionner la fonction (mode VISUAL)
2. Ouvrir les actions : `Espace c a`
3. Choisir **"Fix code"**

Résultat automatique :

```
void process(int* ptr) {
    if (ptr != nullptr) {
        *ptr = 42;
    }
}
```

Le code est **remplacé automatiquement**

4.3 Scénario 3 : Compiler et Corriger Erreurs

Workflow complet :

1. Compiler : `Espace m m`
2. Si erreurs, ouvrir la liste : `Espace m e`
3. Naviguer vers la première erreur : `Espace m n`
4. Corriger l'erreur
5. Aller à l'erreur suivante : `Espace m n`
6. Recompiler : `Espace m m`
7. Si OK, exécuter : `Espace m r`

Chapitre 5

Coefficients de Génération

5.1 Les 5 Coefficients

5.1.1 Temperature (0.0 - 2.0)

Défaut : 0.7
Description : Contrôle la créativité et l'aléatoire des réponses

Valeur	**Comportement**	**Usage**
0.0 - 0.3	Déterministe, précis	Code critique
0.4 - 0.7	Équilibré	Usage général ⋆
0.8 - 1.2	Créatif, varié	Brainstorming
1.3 - 2.0	Très créatif	Expérimentation

TABLE 5.1 – Valeurs de Temperature

5.1.2 Max_Tokens (100 - 8000)

Défaut : 2000
Description : Limite la longueur maximale de la réponse

Valeur	**Type de Réponse**
500	Réponse courte
1000	Réponse moyenne
2000	Réponse longue ⋆
4000	Documentation complète
8000	Très long (coûteux)

TABLE 5.2 – Valeurs de Max_Tokens

5.1.3 Top_P (0.0 - 1.0)

Défaut : 0.9
Description : Contrôle la diversité du vocabulaire

5.1.4 Frequency_Penalty (-2.0 - 2.0)

Défaut : 0.0
Description : Pénalise les mots répétés fréquemment

5.1.5 Presence_Penalty (-2.0 - 2.0)

Défaut : 0.0
Description : Encourage à aborder de nouveaux sujets

5.2 Configurations Recommandées

5.2.1 Code Critique

```
temperature = 0.2
max_tokens = 3000
top_p = 0.8
frequency_penalty = 0.0
presence_penalty = 0.0
```

Résultat : Code déterministe, précis, sûr

5.2.2 Explications Pédagogiques (Défaut)

```
temperature = 0.7
max_tokens = 2000
top_p = 0.9
frequency_penalty = 0.0
presence_penalty = 0.0
```

Résultat : Équilibré, clair, varié

5.2.3 Brainstorming Créatif

```
temperature = 1.2
max_tokens = 2000
top_p = 0.95
frequency_penalty = 0.5
presence_penalty = 0.5
```

Résultat : Créatif, diversifié, original

5.3 Modifier les Coefficients

5.3.1 Pour :Ask (effet immédiat)

```
nvim ~/.local/bin/ask-openrouter
# Modifier lignes 21-25
"temperature": 0.7,
"max_tokens": 2000,
"top_p": 0.9,
```

Sauvegarder avec `:w`

5.3.2 Pour CodeCompanion (relancer Neovim)

```
nvim ~/.config/nvim/init.lua
# Chercher section schema (ligne ~78)
```

Sauvegarder puis **relancer Neovim**

Chapitre 6

Configuration

6.1 Fichiers de Configuration

6.1.1 `~/.config/nvim/init.lua`

Rôle : Configuration principale de Neovim
Contenu :

- Configuration lazy.nvim
- Configuration codecompanion.nvim
- Adaptateur OpenRouter
- System prompt PatIA 120b (3 emplacements)
- Coefficients de génération
- Commandes personnalisées
- Raccourcis clavier

6.1.2 `~/.local/bin/ask-openrouter`

Rôle : Script d'interrogation directe
Contenu :

- Appel curl à OpenRouter API
- Modèle : openai/gpt-oss-120b
- System prompt PatIA 120b
- Coefficients de génération
- Parsing JSON avec jq

6.1.3 `~/.OPENROUTER_API_KEY`

Rôle : Stockage sécurisé de la clé API
Format :

```
export OPENROUTER_API_KEY="sk-or-v1-..."
```

Permissions : 600 (lecture seule propriétaire)

6.2 Plugins Installés

lazy.nvim Gestionnaire de plugins moderne
codecompanion.nvim Intégration IA dans Neovim
tokyonight.nvim Thème de couleurs
nvim-treesitter Coloration syntaxique avancée
telescope.nvim Navigation et recherche

Chapitre 7

Personnalisation

7.1 Changer le Nom de l'Assistant

Fichiers à modifier :

1. `ask-openrouter` (ligne 29)
2. `init.lua` (3 emplacements : lignes 124, 154, 190)

Automatiquement :

```
sed -i 's/PatIA 120b/MonIA/g' ~/.local/bin/ask-openrouter
sed -i 's/PatIA 120b/MonIA/g' ~/.config/nvim/init.lua
```

7.2 Changer le Modèle

Modèles disponibles :

Modèle	Qualité	Coût	Français
openai/gpt-oss-120b	⋆⋆⋆⋆	€	
anthropic/claude-3.5-sonnet	⋆⋆⋆⋆⋆	€€€	
anthropic/claude-3-haiku	⋆⋆⋆	€	
mistralai/mistral-7b-instruct	⋆⋆⋆	Gratuit	

TABLE 7.1 – Modèles recommandés

7.3 Ajouter des Commandes

Template dans `init.lua` :

```
vim.api.nvim_create_user_command("MaCommande", function(opts)
  -- Code Lua ici
  local argument = opts.args
  vim.cmd("!mon-script " .. argument)
end, {
  nargs = "+",
  desc = "Description de ma commande"
})
```

7.4 Ajouter des Raccourcis

Template :

```
vim.api.nvim_set_keymap("n", "<F4>", ":MonAction<CR>",
  { noremap = true, silent = false, desc = "Mon action" })
```

Chapitre 8

Dépannage

8.1 Problème : Neovim $<$ 0.11

Erreur

```
CodeCompanion.nvim requires Neovim 0.11+
```

Solution

```
bash fix_neovim_version.sh
```

Choisir **option 1** (AppImage)

8.2 Problème : :Ask ne fonctionne pas

Cause : API Key non chargée
Solution :

```
# Verifier
cat ~/.OPENROUTER_API_KEY
echo $OPENROUTER_API_KEY

# Si vide, charger
source ~/.OPENROUTER_API_KEY
```

8.3 Problème : Réponses en anglais

Cause : CodeCcompanion ne transmet pas toujours le system prompt
Solution : Utiliser `:Ask` au lieu de CodeCompanion

8.4 Diagnostic Complet

Script de Diagnostic

```
bash verify_installation.sh
```

Effectue 7 sections de tests :

1. Dépendances système
2. Neovim (version, config)
3. API Key (validité, connexion)
4. Configuration Neovim
5. Script ask-openrouter
6. Tests fonctionnels réels
7. Documentation

Chapitre 9

Architecture Technique

9.1 Flux de Données - :Ask



FIGURE 9.1 – Architecture de la commande :Ask

9.2 System Prompt - Structure

3 emplacements dans init.lua :

1. **opts.system_prompt** (ligne ~124)
2. **prompts["Custom Prompt"]** (ligne ~154)
3. **system_prompt principal** (ligne ~190)

Contenu :

— Identité (PatIA 120b)
— Règle de langue (FRANÇAIS obligatoire)
— Langage par défaut (C++11)
— Expertises
— Principes
— Format des réponses

Chapitre 10

Référence Rapide

10.1 Commandes Essentielles

```
:Ask <question>                    " Question en francais
:AskBuffer                         " Envoyer fichier au chat
:Generate <prompt>                 " Generer du code
:Translate <langage>               " Traduire code
:MakeRun                           " Compiler et executer
```

10.2 Raccourcis Essentiels

```
Espace c c                         " Chat interactif
Espace c a                         " Actions sur selection
Espace c s                         " Envoyer message (Normal)
Ctrl+s                             " Envoyer message (Insert)
Espace m m                         " Compiler (make)
Espace m r                         " Make & Run
F2                                 " Script compilation custom
F3                                 " Script sauvegarde custom
```

10.3 Coefficients par Défaut

```
temperature         : 0.7
max_tokens          : 2000
top_p               : 0.9
frequency_penalty   : 0.0
presence_penalty    : 0.0
```

10.4 Fichiers Importants

~/.config/nvim/init.lua Config Neovim
~/.local/bin/ask-openrouter Script :Ask
~/.OPENROUTER_API_KEY Clé API

10.5 Scripts Utiles

```
bash install.sh                    # Installer
bash verify_installation.sh        # Verifier
bash uninstall.sh                  # Desinstaller
bash fix_neovim_version.sh         # Corriger Neovim < 0.11
```

10.6 Tests Rapides

```
# Terminal
ask-openrouter "quel est ton nom ?"

# Neovim
nvim
:Ask bonjour
```

Annexes

A. Modifications Récentes

Ajout ligne 126 et 160 du system prompt :

```
Ne repete pas la question en anglais
```

Ces modifications empêchent PatIA de reformuler la question en anglais avant de répondre.

B. Modèles Alternatifs

Pour code de qualité :

— anthropic/claude-3.5-sonnet

Pour rapidité :

— anthropic/claude-3-haiku

— google/gemini-pro

Pour gratuit :

— meta-llama/llama-3.1-70b-instruct

— mistralai/mistral-7b-instruct

À éviter :

— openai/gpt-4o-mini (refuse le français)

C. Ressources

— **OpenRouter** : https://openrouter.ai/

— **Neovim** : https://neovim.io/

— **CodeCompanion** : https://github.com/olimorris/codecompanion.nvim

— **lazy.nvim** : https://github.com/folke/lazy.nvim

— **Documentation locale** : `/root/IA/NEOVIM/`

D. Crédits

Configuration Claude Code

Date 2025-11-01

Version 1.0

Licence Configuration libre, outils avec leurs licences respectives

PatIA 120b

Documentation Complète v1.0

Assistant IA Expert C++11 pour Neovim